



ESCUELA DE INGENIERÍA EN COMPUTACIÓN  
PROGRAMA DE MAESTRÍA EN COMPUTACIÓN

---

# Identification of primary (P) and secondary (S) waves in time series of seismic events using Convolutional Neural Networks

---

Thesis

Presented in partial fulfilment of the requirements to opt for the degree  
of *Magister Scientiæ* in Computer Science

Author

Irene Gamboa Padilla

Thesis Director

Francisco J. Torres Rojas, Ph.D.

July 2021



*This thesis is dedicated to everyone who has helped me in this journey; especially my parents Virgita and Marco, my sister Jessica, my niece Priscila, and my partner and best friend Andrés, who have always inspired me to achieve my goals.*

---

*Esta tesis está dedicada a todos los que me han ayudado en esta aventura; especialmente a mis padres Virgita y Marco, a mi hermana Jessica, mi sobrina Priscila, y mi compañero y mejor amigo Andrés, quienes siempre me han inspirado a lograr mis metas.*

# Acknowledgment

I want to give thanks to my advisor, *Francisco Torres*, for all the help that he provided throughout my studies, not only as an advisor but also as my professor and friend throughout my university career. I also want to thank my *reviewers and other members of the committee* for their countless comments and feedback during the whole process of doing this thesis, to the research group *Happy Few*, who guided and helped me through the entire process and its member *Andrés Morales Esquivel*, who helped me with the statistical analysis of the results of this thesis and the pre-processing data investigation, to *Northern California Earthquake Data Center (NCEDC)* at the *Berkeley University* for gathering and providing the Seismic Data Set and for providing. Finally, I would like to thank *my family and friends* for all the support and patience that they provided me during this adventure.

## **Abstract**

In the last five years, the fundamental problem of seismology, which consists of the location in 4 dimensions (three spatial dimensions and one temporal) of the seismic source, has become one of the most essential and complicated challenge in this area.

Modern methods of advanced computing, such as machine learning for signal processing, pattern recognition, and data mining, have proven to be a promising and necessary path in different areas of research, particularly in seismology. Recent works offer a preliminary solution to the problem mentioned above, the adaptation and execution of algorithms that allow identifying the arrival of seismic waves (and therefore the creation of catalogs), is in an initial phase.

In the present work, we intend to adapt, compare different “Convolutional Neural Networks” with different hyper-parameters and pre-processing steps for the identification of the arrival of seismic body waves: the primary wave ( $P$ ) and the secondary wave ( $S$ ), using machine learning, which is the base process that will allow scientists to learn what is happening inside of the Earth.

## ACTA DE APROBACION DE TESIS

Identification of primary (P) and secondary (S) waves in time series of seismic events  
using Convolutional Neural Networks

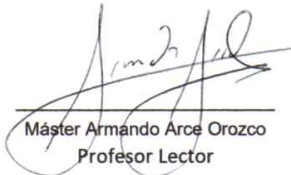
### TRIBUNAL EXAMINADOR



Doctor Francisco Torres Rojas  
Profesor Asesor



Doctor Roberto Cortés Morales  
Profesor Lector



Master Armando Arce Orozco  
Profesor Lector



Master Jose Andrés Mena Arias  
Lector Externo



Dra. Lilliana Sancho Chavarría  
Coordinadora  
Unidad de Posgrado, Escuela de Computación

31 de julio 2020



---

# Content

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Theoretical Framework</b>	<b>7</b>
2.1	Seismological Concepts . . . . .	8
2.1.1	Primary Wave ( $P$ ) . . . . .	8
2.1.2	Secondary Wave ( $S$ ) . . . . .	9
2.1.3	Seismograms . . . . .	10
2.2	Time Series . . . . .	10
2.3	Classification Models . . . . .	11
2.3.1	Convolutional Neural Network . . . . .	11
2.3.2	Dense Neural Network . . . . .	16
2.3.3	Evaluation Metrics . . . . .	17
2.4	Design of Experiments ( $DoE$ ) . . . . .	19
2.5	Analysis of Variance ( $ANOVA$ ) . . . . .	19
2.6	Related Work . . . . .	21
2.6.1	PhaseNet . . . . .	21
2.6.2	AR Picker . . . . .	22
2.6.3	Fingerprint And Similarity Thresholding ( $FAST$ ) . . . . .	23
2.6.4	ConvNetQuake . . . . .	24
<b>3</b>	<b>Research Methodology</b>	<b>26</b>
3.1	Hypothesis . . . . .	27
3.2	Objectives . . . . .	28
3.2.1	Main Objective . . . . .	28
3.2.2	Specific Objectives . . . . .	28
3.3	Scope and Limitations . . . . .	28
<b>4</b>	<b>Preprocessing Seismic Wave Data</b>	<b>30</b>
4.1	Sliding Windows . . . . .	31
4.2	Scaling . . . . .	32

4.3	Balancing . . . . .	33
4.4	Pre-processing . . . . .	33
<b>5</b>	<b>Experimentation</b>	<b>36</b>
5.1	Experimentation Setting . . . . .	37
5.1.1	Data Set . . . . .	37
5.1.2	Pre-Experiments . . . . .	38
5.1.3	Final Experiments . . . . .	50
5.1.4	Preprocessing Seismic Wave Data Implementation . . . . .	51
5.2	Experimentation Execution . . . . .	56
5.2.1	Results . . . . .	58
<b>6</b>	<b>Discussion</b>	<b>64</b>
6.1	Net . . . . .	64
6.2	Scaler . . . . .	66
6.3	Window . . . . .	67
6.4	Network-Scaler . . . . .	68
6.5	Network-Window . . . . .	71
6.6	Window-Scaler . . . . .	73
6.7	Network-Scaler-Window . . . . .	75
6.8	Summary . . . . .	79
6.8.1	Final Results . . . . .	81
<b>7</b>	<b>Conclusions and Future Work</b>	<b>85</b>
7.1	Conclusions . . . . .	85
7.2	Future Work . . . . .	86
7.3	Final Words . . . . .	87
	<b>References</b>	<b>88</b>
	<b>Acronyms</b>	<b>92</b>
	<b>Annexes</b>	<b>93</b>

---

## List of figures

---

1.1	Tectonic Plates [30] . . . . .	2
1.2	Waves $P$ (purple) and $S$ (red) . . . . .	4
1.3	Number of Earthquakes per year (worldwide) [17] . . . . .	5
2.1	Primary Wave [4] . . . . .	9
2.2	Secondary Wave [4] . . . . .	10
2.3	Classic Seismogram [3] . . . . .	10
2.4	Digital Seismogram . . . . .	10
2.5	Convolutional Neural Network Example [10] . . . . .	12
2.6	Le-Net architecture [16] . . . . .	16
2.7	Dense Neural Network Representation on TensorFlow Playground [37] .	16
2.8	PhaseNet Architecture [52] . . . . .	21
2.9	Feature extraction steps in FAST [51] . . . . .	23
2.10	ConvNetQuake Architecture [32] . . . . .	24
4.1	Sliding Window example . . . . .	32
4.2	Pre-processing and training flow . . . . .	34
4.3	Pre-processing and prediction flow . . . . .	35
5.1	Pre-Experiment Convolutional - Normal Q-Q plot. . . . .	43
5.2	Pre-Experiment Convolutional - Histogram plot . . . . .	43
5.3	Pre-Experiment Dense - Normal Q-Q plot. . . . .	49
5.4	Pre-Experiment Dense - Histogram plot . . . . .	49
5.5	Graph showing the difference of accuracy per layer levels in Dense model	50
5.6	Normal Q-Q plot – $P$ wave. . . . .	60
5.7	Normal Q-Q plot – $S$ wave. . . . .	60
5.8	Residuals vs Fitted – $P$ wave. . . . .	61
5.9	Residuals vs Fitted – $S$ wave. . . . .	61
6.1	Graph showing the difference of score per network levels in the $P$ -wave	65
6.2	Graph showing the difference of score per network levels in the $S$ -wave	65



6.3	Graph showing the difference of score per scaler levels in the <i>P-wave</i> . .	66
6.4	Graph showing the difference of score per scaler levels in the <i>S-wave</i> . .	67
6.5	Graph showing the difference of score per window levels in the <i>P-wave</i>	68
6.6	Graph showing the difference of score per window levels in the <i>S-wave</i> .	68
6.7	Graph showing the difference of score per network and scaler levels in the <i>P-wave</i> . . . . .	69
6.8	Graph showing the difference of score per network and scaler levels in the <i>S-wave</i> . . . . .	71
6.9	Graph showing the difference of score per network and window levels in the <i>P-wave</i> . . . . .	72
6.10	Graph showing the difference of score per network and window levels in the <i>S-wave</i> . . . . .	72
6.11	Graph showing the difference of score per window and scaler levels in the <i>P-wave</i> . . . . .	74
6.12	Graph showing the difference of score per window and scaler levels in the <i>S-wave</i> . . . . .	75
6.13	Graph showing the difference of score per network, scaler and window levels in the <i>P-wave</i> . . . . .	78
6.14	Graph showing the difference of score per network, scaler and window levels in the <i>S-wave</i> . . . . .	79
6.15	Difference seconds between real and predicted <i>P-wave-1</i> . . . . .	81
6.16	Difference seconds between real and predicted <i>P-wave-2</i> . . . . .	82
6.17	Difference seconds between real and predicted <i>P-wave-3</i> . . . . .	82
6.18	Difference seconds between real and predicted <i>S-wave-1</i> . . . . .	83
6.19	Difference seconds between real and predicted <i>S-wave-2</i> . . . . .	83
6.20	Difference seconds between real and predicted <i>S-wave-3</i> . . . . .	84

---

## List of Tables

---

2.1	True Positive + False Positive = Total Predicted Positive . . . . .	17
2.2	True Positive + False Negative = Actual Positive . . . . .	17
5.1	Table with Factors and Levels of Pre-Experiments for Conv Net . . . . .	39
5.2	Table with Average Accuracy Results for Pre-Experiments . . . . .	42
5.3	Table with Factors and Levels of Experiments of $P$ Seismic Wave for Dense Net . . . . .	46
5.4	Table with Factors and Levels of Experiments . . . . .	51
5.5	Table with Levene Results of $P$ and $S$ Seismic Wave . . . . .	61
6.1	Differences of Means Summary . . . . .	80
6.2	Differences of Means Summary – $P$ -wave . . . . .	80
6.3	Differences of Means Summary – $S$ -wave . . . . .	80
6.4	Average of Differences Between Real and Predicted . . . . .	81

---

## Listings

---

5.1	Convolutional Neural Network with Layer Parameter Implementation .	39
5.2	Convolutional Neural Network with Layer Parameter Summary . . . . .	40
5.3	Convolutional Neural Network Two Implementation . . . . .	44
5.4	Convolutional Neural Network Two Summary . . . . .	45
5.5	Dense Network Implementation . . . . .	46
5.6	Dense Network Summary . . . . .	47
5.7	Download Data from <b>Northern California Earthquake DataCenter</b>	52
5.8	Generation Files . . . . .	54
5.9	Preprocessing Data . . . . .	55
5.10	Run Experiments . . . . .	57
5.11	Calculate F1 Score . . . . .	58
5.12	Levene Test . . . . .	61
5.13	ANOVA - Experiments P Seismic Wave . . . . .	62
5.14	ANOVA - Experiments S Seismic Wave . . . . .	62
6.1	Tukey - Network-Scaler – <i>P-wave</i> . . . . .	70
6.2	Tukey - Network-Scaler – <i>S-wave</i> . . . . .	70
6.3	Tukey - Window-Scaler – <i>P-wave</i> . . . . .	73
6.4	Tukey - Window-Scaler – <i>S-wave</i> . . . . .	74
6.5	Tukey - Network-Window-Scaler – <i>P-wave</i> . . . . .	76
6.6	Tukey - Network-Window-Scaler – <i>S-wave</i> . . . . .	77
1	Python Code - Analyze <b><i>P-wave</i></b> . . . . .	93
2	Python Code - Analyze <b><i>S-wave</i></b> . . . . .	94
5	Average difference calculation . . . . .	97
6	Final Results Generation . . . . .	99

## Chapter 1

---

### Introduction

---

Costa Rica is part of the volcanic arc of Central America, formed by the process of subduction of the Coco plate below the Caribbean and the microplate of Panama along the Mesoamerican trench. Towards the south of the continent, the Panama microplate and the Coco plate interact with the Nazca plate, forming a triple point and a transformation fault with dextral displacement known as the Panama fracture zone (see Figure 1.1) [1]. Due to the tectonic complexity of the region, Costa Rica is a highly seismic country, with the potential to generate large-scale events ( $M > 7.0$ ) in the subduction zones below the Nicoya and Osa peninsulas and in the Caribbean region, where the Caribbean plate is subducting below the microplate of Panama.

The seismic activity in Costa Rica began to be documented from 1608 until 1910 by Cleto Gonzáles Viquez [45]. However, these data make a very general description of the superficial manifestation of the events, and although they constitute an invaluable record, the physical characterization of the seismic source is very limited. It was not until 1888 that seismological studies began in Costa Rica, generating the first “National Seismological Bulletin” in 1901 [29].



**Figure 1.1:** Tectonic Plates [30]

---

At the beginning of the 1980s, two organizations that carry out the country's seismological observation were created in Costa Rica, in 1982 the National Seismological Network known as **RSN UCR-ICE** (University of Costa Rica) [44] which works in conjunction with the Costa Rican Institute of Electricity (ICE). In 1984 the Vulcanological Observatory of Costa Rica known as **OVSICORI** (National University) [28].

Currently, the RSN is made up of about 210 stations throughout the country that register in real-time in the seismology laboratory of the Central American School of Geology of the University of Costa Rica [44].

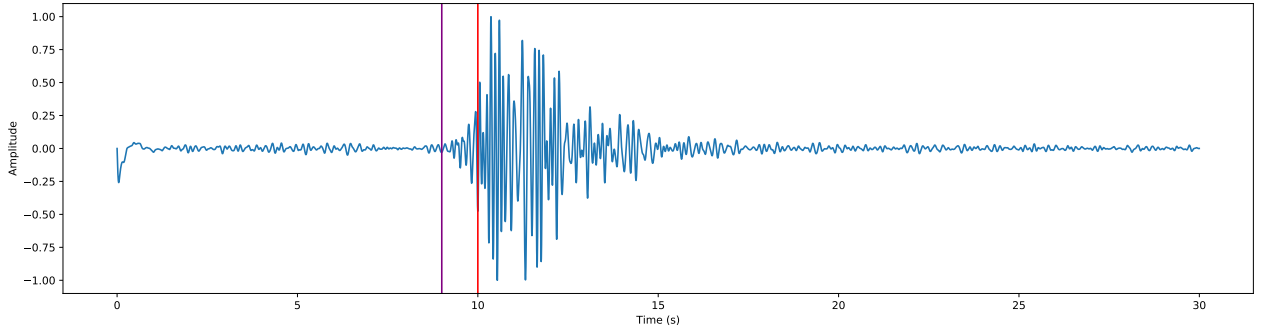
Additionally, the OVSICORI has at least 70 stations throughout the country. Data transmission occurs in real-time to the observatory through the internet. [50]. They claim to have the densest seismological network in Latin America [7].

At the beginning of this investigation, it was proposed to carry out interdisciplinary work together with the OVSICORI-UNA and use data from Costa Rica for it. However, between the OVSICORI-UNA and any other entity or person, there must be an intermediary, who can extract the data and prepare them in such a way that they can be used in a suitable format so that they can be used in this research. To be able to process them, an expert (seismologist) is needed, who must take from his work time to carry out the process, which does not facilitate a fluid process in the delivery of the data.

This investigation began with data provided by the OVSICORI-UNA, 285 seismic events, but for its usage in deep learning, such as the training of convolutional neural network, it did not contain a very large sample of events and caused over-fitting. Therefore, we decided to use to a more open and public catalog of seismic events (see figure 1.2.

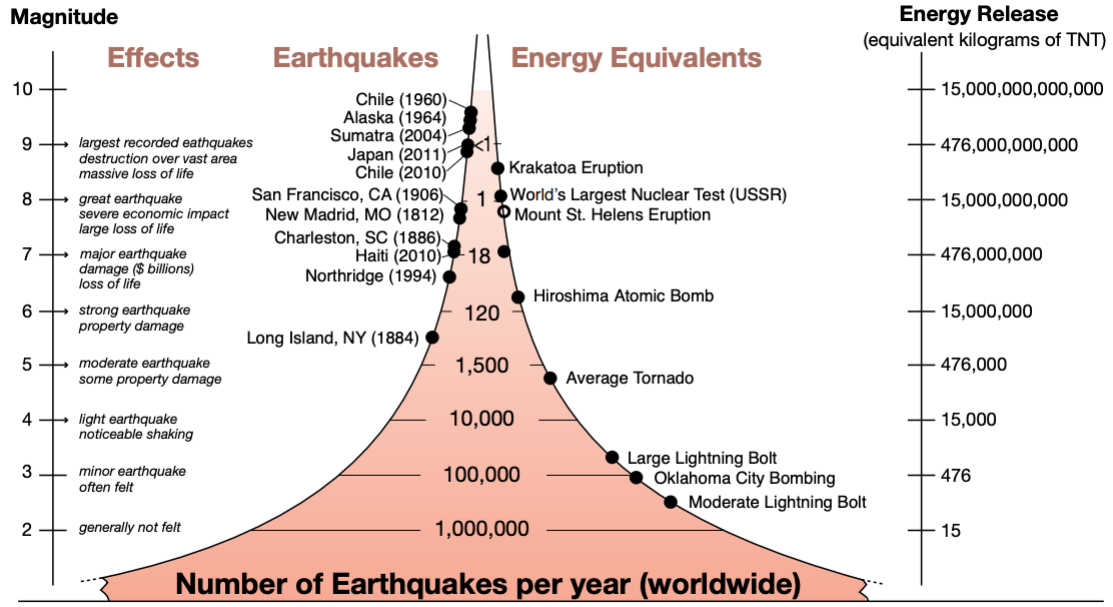
---

In this research, we used the information compiled by the **Northern California Earthquake Data Center - UC Berkeley Seismic Laboratory** from **llenar** to **llenar**, and the use of automatic learning algorithms, to develop a proposal that allows us to detect the arrival of primary wave ( $P$ ) and the secondary wave ( $S$ ) in time series of seismic events.



**Figure 1.2:** Waves  $P$  (purple) and  $S$  (red)

In the last five years, the fundamental problem of seismology consists in locating in 4 dimensions (three spatial dimensions and one temporal) the seismic source, this has become one of the most essential and complicated challenges in this area [15].



**Figure 1.3:** Number of Earthquakes per year (worldwide) [17]

The increasing amount of seismic data (see Figure 1.3) due to the increase of the number of sensors that can gather information from earthquakes across the globe makes it hard to manually identify the  $P$  and the  $S$  waves for all the events that happen daily [17; 52]. While methods such as AR-picker, PhaseNet, ConvNetQuake, and others have attempted to automatically identify these waves, they are not exact and can be improved [2; 51; 32; 52].

The success that has been obtained in different artificial intelligence projects to learn how to catalog data such as identifying objects in images, classification of extra-solar planets [13], classification of heart rate time series [39]; has facilitated the development of research and improvements in the use of machine learning algorithms for the classification of different types of data. That is why machine learning algorithms are chosen for the development of this research.



---

This document is divided as follows: The first chapter is the description of the problem, in the second chapter summary of the most important concepts that the proposed project will seek to address is provided. The third chapter contains the objectives and hypotheses. In chapter four we present the description of the pre-processing data, in chapter five, provide the experimentation setting, contain the description for the setup of the experimentation, what factors and levels will be analyzed, the pre-processing required for the dataset, the architecture of the neural networks. In chapter six we will present the results of running the described experiments, and finally, in chapter seven we will discuss the results and present possible future work for this thesis.

---

## Theoretical Framework

---

## 2.1 Seismological Concepts

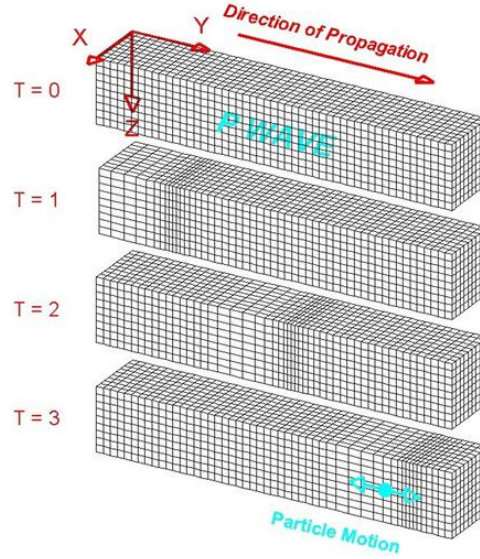
Seismology is the area of geophysics that is responsible for the study of earthquakes and the propagation of seismic waves, part of this study includes determining the hypo-centers, the location and the time the earthquake occurred [42]. As the main objectives of seismology we find:

- Know the internal structure of the Earth, through the study of waves propagation.
- Earthquake origin.
- Seismic damage prevention.

An earthquake or a tremor is generated by the sudden rupture of the earth causing seismic waves; these waves are the energy that travels through the earth. This energy is recorded in the seismometers as seismic-grams, there are several types of waves, but the primordial waves are the waves  $P$  and  $S$  [49].

### 2.1.1 Primary Wave ( $P$ )

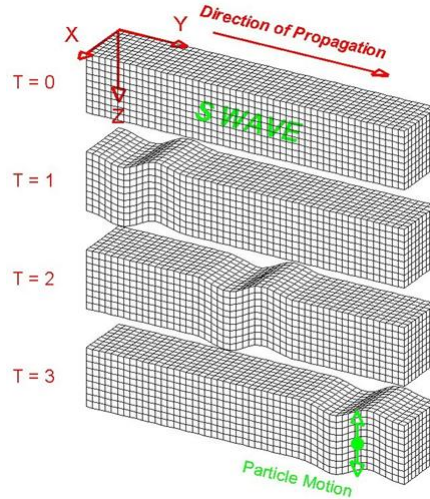
It is known as the primary wave to the fastest seismic wave, and therefore the first to reach the seismic station. The  $P$  waves are also known as compression waves, due to the thrust and pull they make [49]. The particles move in the same direction as the wave is moving, allowing to generate the wave  $P$ , is the direction in the energy is moving and is sometimes called “direction of propagation of the wave” (see Figure 2.1). In Costa Rica the waves  $P$  have an average speed of  $7km/s$  [50].



**Figure 2.1:** Primary Wave [4]

### 2.1.2 Secondary Wave ( $S$ )

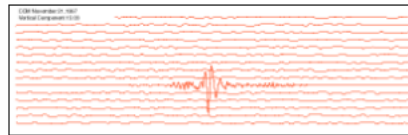
It is known as a secondary wave or cut wave to the second type of wave that is perceived in an earthquake, this is slower than the wave  $P$ ; these waves move the particles up and down, or from side to side perpendicular to the direction of propagation of the wave (see Figure 2.2) [49] In Costa Rica the waves  $S$  have an average speed of  $3.5km/s$  [50].



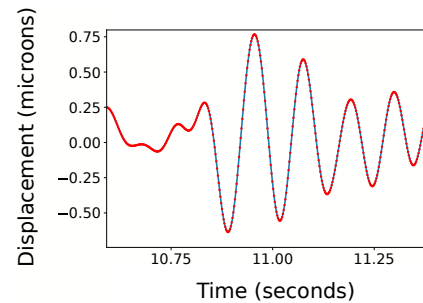
**Figure 2.2:** Secondary Wave [4]

### 2.1.3 Seismograms

A seismogram is a record of soil movement detected by a seismometer [33]. The energy measured in a seismogram results from natural sources, instrumental or anthropogenic noises. In the past, seismograms recorded in rotating paper drums (see Figure 2.3). Currently, virtually all seismographs record information digitally (see Figure 2.4).



**Figure 2.3:** Classic Seismogram [3]



**Figure 2.4:** Digital Seismogram

## 2.2 Time Series

Time series are a serie of indexed data points (or lists or graphs) in order of time. More commonly, a time series is a sequence of data taken at successive points equally

spaced in time. Therefore, it is a discrete-time data stream [48].

Time series are plotted very frequently through line graphs. They are used in statistics, signal processing, pattern recognition, econometrics, mathematical finance, climate prediction, earthquake prediction, electroencephalography, control engineering, astronomy, communications engineering and in any domain of applied science and engineering that involves temporary measurements [41].

## 2.3 Classification Models

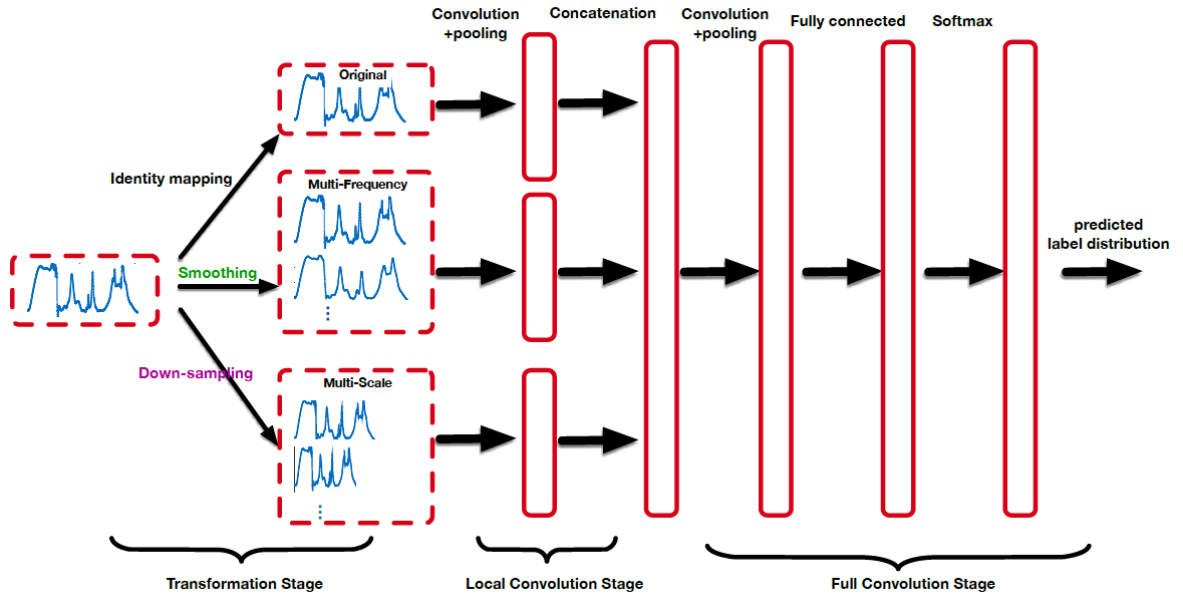
Automatic learning or machine learning is a branch of the Artificial Intelligence (*AI*), which has as its primary objective to develop techniques that allow computers to learn [47]. The idea is to create programs capable of generalizing behaviors from information provided in the form of an example, it is considered as a previous training.

Machine learning has a wide range of applications, including search engines, medical diagnostics, fraud detection in the use of credit cards, stock market analysis, classification of DNA sequences, recognition of speech and written language, games and robotics, among others [11].

### 2.3.1 Convolutional Neural Network

The Convolutional Neural Networks are a category of neural networks, being part of the automatic learning algorithms, they are useful in areas such as recognition and classification of images. They consist of multiple layers of convolutional filters of one or more dimensions, after each layer a function is added and this performs a mapping (see Figure 2.6).

At the beginning is the phase of extraction of characteristics and at the end of the network there is the phase to perform the final classification on the characteristics extracted. In the phase of extraction of characteristics, we find alternating layers of convolutional neurons and reduction of sampling. According to the process and progress of the data throughout this phase, the dimensionality of the data is reduced, this causes that in the further layers the perturbations in the input data are less sensitive, but at the same time the data have more complex characteristics [25].



**Figure 2.5:** Convolutional Neural Network Example [10]

### 2.3.1.1 Hyperparameters

A hyperparameter is a parameter whose value is set before the learning process begins. On the contrary, the values of other parameters are derived through training [8].

To choose a set of optimal hyperparameters for a learning algorithm, different methods are used, for example:

- **Grid Search**

It is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model used, specific parameters are necessary. Grid Search will build a model in each possible parameter combination. It iterates through each combination of parameters and stores a model for each combination [21].

- **Random Search**

It is a direct search method since it does not require derivatives to search for a continuous domain. This basic approach is related to techniques that provide small improvements, such as Directed Random Search and Adaptive Random Search [5].

- **Bayesian Optimization**

It is an approach to optimize objective functions that take a lot of time (minutes or hours) to evaluate. Since the objective function is unknown, the Bayesian strategy is to treat it as a random function and place a prior on it, which is to capture the behavior of the function, then collect the evaluations of functions, which are treated as data, updated the previous one to form the posterior distribution on the objective function. and this distribution, in turn, is used to build an acquisition function that determines what the next inquiry point should be [23].

The following sections explain some of the hyperparameters that are used in the development of the convolutional neural network model.

**2.3.1.1.1 Learning rate** The learning proportion controls how to update the weight in the optimization algorithm [19]. It is responsible for the main learning characteristic and should be chosen in such a way that it is not too high in which we can not converge to the minimum and is not so low that we can not accelerate the learning process.



The definition of the value of the learning proportion depends very much on the optimizer used, and it is recommended to use adaptive learning proportion algorithms [35].

**2.3.1.1.2 Number of epochs** The epochs are the number of times with which the convolutional neural network is training with the same set of training data, the number of times must be increased until a small margin is seeing between the test error and the training error [19].

A large number of times can over-conform to the data and may have generalization problems in the test and the validation set, and they could also cause degradation problems. A smaller number of epochs may limit the potential of the model [35].

**2.3.1.1.3 Batch Size** The batch size is indicative of the number of patterns that will be sample before it is updated the network. If it is of small size, the patterns would be repeated less, which means that the weights are everywhere and the convergence becomes more difficult, on the other hand, if the size is high the learning becomes slow, since after many iterations will change the batch size. It is recommended to test the batch sizes in powers of 2 according to the size of the data [35].

**2.3.1.1.4 Activation function** The activation function is a function that introduces non-linearity in the model. Defines the output of a node given an input or set of inputs [19].

**2.3.1.1.5 Number of hidden layers and units** The hidden layers are layers between the input layers and the output layers, where the artificial neurons take a set of weighted inputs and produce an output through an activation function. It is good to add more layers until the test error no longer improves. Having a small number of

units can lead to failures while having more units is not usually harmful with proper regularization [19].

However, a very high number can present problems such as excessive adjustment, disappearance, and an explosion of gradient problems, and a lower number can cause a model to have high bias and a low potential. It depends a lot on the size of the data used for training [35].

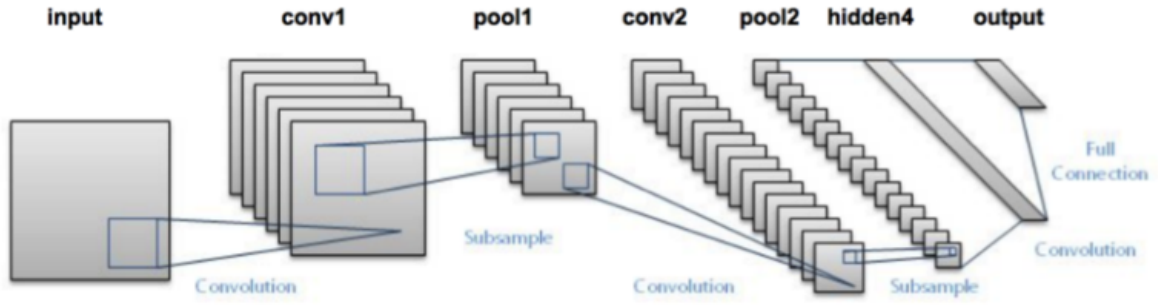
**2.3.1.1.6 Filter Size** The filter is a matrix of weights with which it moves in the input, these weights in the filter matrix are derived while training the data. It provides a measure of how close an input piece looks to a feature [36].

The smaller filters collect as much local information as possible, the larger filters represent more global, high-level and representative information. It should be taken into account that filters with odd sizes are generally used.

**2.3.1.1.7 Number of Channels** It is also known as the number of color channels for this input, in later stages, it is equal to the number of filters used for the convolution stage. The larger the number of channels, the more filters used, the more functions learned, and the more possibilities there are for an excessive adjustment and vice versa [36].

### 2.3.1.2 Le-Net

In 1998 Yann LeCun et al proposed a new convolutional neural network structured called Le-Net in general, refers to lenet-5 it is a simple convolutional neural network [20]. The Le-Net architecture consists of two sets of convolutional, activation, and pooling layers, followed by a fully-connected layer, activation, another fully-connected, and finally a softmax classifier.

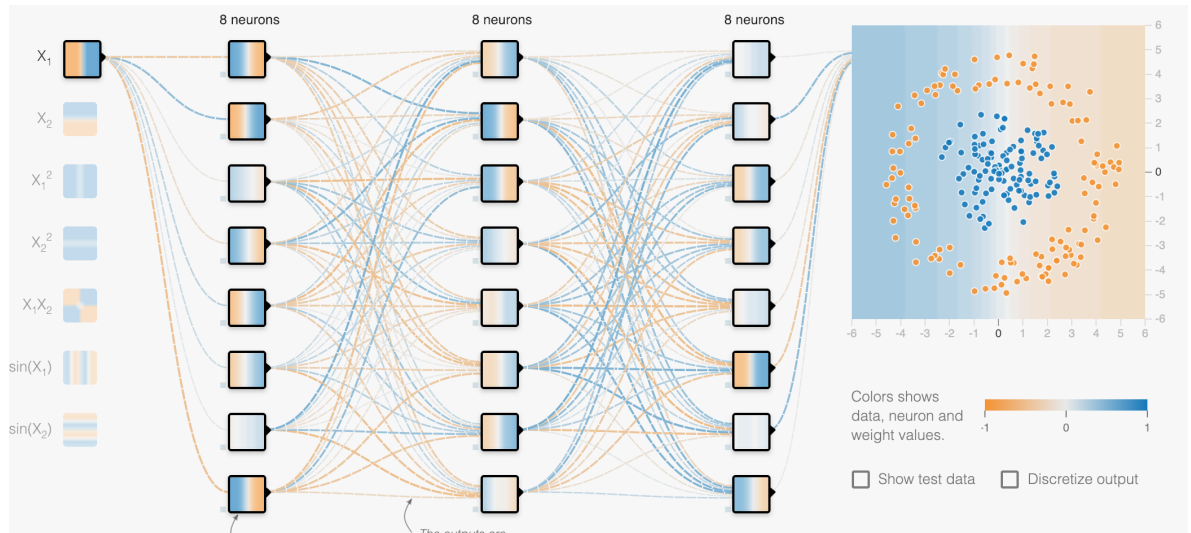


**Figure 2.6:** Le-Net architecture [16]

### 2.3.2 Dense Neural Network

The Dense Neural Networks are a category of neural networks. The idea for this category is to have layers are fully connected by the neurons in a network layer [37].

Each layer receives the input from the layers present in the previous layer (2.7), so this are densely connected.



**Figure 2.7:** Dense Neural Network Representation on TensorFlow Playground [37]

### 2.3.3 Evaluation Metrics

#### 2.3.3.1 Precision

This tells us how accurate is a model. It is a good measure to determine when False Positive costs are high. [43].

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}}$$

		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

**Table 2.1:** True Positive + False Positive = Total Predicted Positive

#### 2.3.3.2 Recall

It allows us to calculate how many of the positives found by the model are real. It is a metric that is used to determine if there is a high cost associated with False Negative [43].

$$\text{Recall} = \frac{\text{TruePositive}}{\text{FalseNegative} + \text{TruePositive}}$$

		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

**Table 2.2:** True Positive + False Negative = Actual Positive

### 2.3.3.3 F1 Score

F1 Score is used to find a balance between **Precision** and **Recall** [43].

$$\text{F1 Score} = 2 * \frac{\textit{Precision} * \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

## 2.4 Design of Experiments (*DoE*)

The *DoE* is a systematic and rigorous approach to solving engineering problems that applies principles and techniques to a collection of data to ensure a valid, defensible and supportable conclusion. In addition to the above, this model is carried out under the condition of minimizing resource expenditure, runs of experiments, time and money [24].

There are four types of general problems in which it is useful to apply design of experiments:

1. **Comparative:** the researcher is interested in knowing if the change of a factor generates improvement in a process.
2. **Characterization:** it is desired to know what are the factors that affect a process to be able to sort them according to their level of impact.
3. **Modeling:** It is of interest to model a process whose output is a mathematical function with high predictive power and have a good estimate of the coefficients of said function.
4. **Optimization:** the engineer wants to determine the optimal parameters of the factors of a process. That is, determine for each factor the level that optimizes the response of the process.

## 2.5 Analysis of Variance (*ANOVA*)

The *ANOVA* makes it possible to ensure that the variation in the results of an experiment is not greater than the sum of the variations of the factors and a certain degree of error in their measurements. This allows accepting or rejecting the hypothesis with an error probability (preferably very low) based on statistical evidence [46, p507].

The purpose of *ANOVA* is to analyze the relationship between a quantitative variable  $X$  and a qualitative variable  $Y$  of  $k$  attributes. Each attribute  $i$  defines a population given by the quantitative variable [38, p247].

$X_i$  : variable  $X$  restricted to attribute  $i$ .

Thus, have  $k$  populations  $X_1, X_2, \dots, X_n$  (called treatments) assumed to be normal, independent, with similar variances and population means  $\mu_1, \mu_2, \dots, \mu_n$ . We want to determine if  $X$  does not vary according to the attribute of  $Y$ , that is, if the populations are equivalent and then the treatments are equally effective. For this, the hypotheses are proposed and contrasted:

$H_0$  :  $X$  does not vary according to the attribute of  $Y$  (equivalent populations, that is  $\mu_1 = \mu_2 = \dots = \mu_k$ ).

$H_1$  :  $X$  varies according to the attribute of  $Y$  (non-equivalent populations), at least two of the means are not equal.

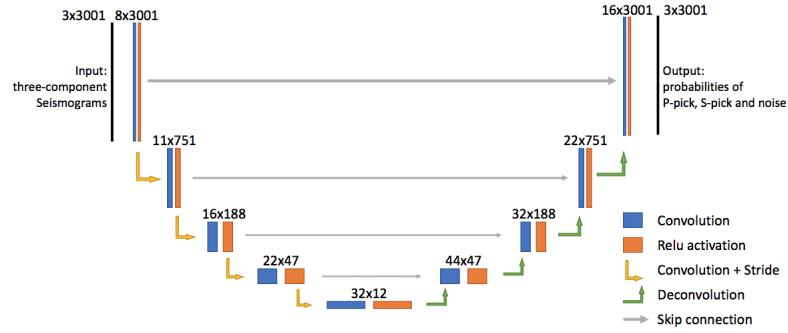
A great quality of ANOVA is that it allows to analyze several populations, while other methods do not allow more than two.

The selected *software* tool to run the variance analysis process and present the data graphically consists of the package provided by the project R [34].

## 2.6 Related Work

### 2.6.1 PhaseNet

PhaseNet is a method of selecting the arrival time of the waves  $P$  and  $S$  by means of a Convolutional Neural Network (see Figure 2.8), uses seismic waveforms of three components as input and generates probability distributions of a point being the wave  $P$ , the wave  $S$  or noise as output [52]. It was designed by the geophysicists Weiqiang Zhu and Gregory C. Beroza of Stanford University.



**Figure 2.8:** PhaseNet Architecture [52]

The data set that they use are from “Northern California Seismic Network”, it has data from 580 stations in northern and central California and 159 stations in southern California [6], it contains more than seven million samples of waveforms extracted from more than thirty years of earthquake recordings.

In [52] authors show that PhaseNet can detect and select the arrivals of the waves  $P$  and  $S$  effectively within waveforms of known seismic events. The F1 score for the detection of the waves  $P$  is 0.896, and for the waves  $S$  is 0.801.



### 2.6.2 AR Picker

It is a technique capable of capturing the times of the automatic start of the waves  $P$  and  $S$  in registers of strong movements, using a simple comparative approach [2]. Use the method Auto Regression- Akaike Information Criterion ( $AR-AIC$ ) and the relationship Short-Term-Average ( $STA$ )/Long-Term-Average ( $LTA$ ). For this technique were used data from observation networks in Japan.

The procedure for detecting the start time of the waves  $P$  and  $S$  contains the following steps. The first 5 steps are applied to estimate the start time of the wave  $P$  and the last 4 steps are applied to estimate the start time of the wave  $S$ .

1. Generation of a “cumulative envelope” function of the vertical acceleration register.
2. Calculation of the relationship  $STA/LTA$  for the envelope function.
3. First application of the method  $AR-AIC$ .
4. Second application of the method  $AR-AIC$ .
5. Estimation of the start time of the wave  $P$ .
6. Calculation of the  $STA-LTA$  of the predominant component of the horizontal acceleration register.
7. Calculation of  $STA-LTA$  using the inverse time direction.
8. Application of the method  $AR-AIC$ .
9. Estimation of wave start time  $P$ .

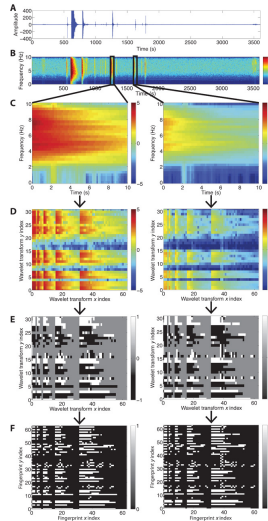
The important characteristic of this technique is the ability to delimit the detection intervals using only the maximum or minimum values. This simplification limits the

detection rate when applied to all or any observation record (mainly when applied to records at regional distances of 100 km and more) [2].

### 2.6.3 Fingerprint And Similarity Thresholding (*FAST*)

*FAST* is a method that allows you to detect earthquakes using waveform similarity, and you can analyze a week of continuous seismic waveform data in less than 2 hours. It adapts a data mining algorithm, originally designed to identify similar audio clips within large databases [51].

First, they create “fingerprints” of waveforms by extracting discriminative features (see Figure 2.9), then it group the similar “fingerprints” into a database, to facilitate search, and finally it generates a list of earthquake detections.



**Figure 2.9:** Feature extraction steps in FAST [51]

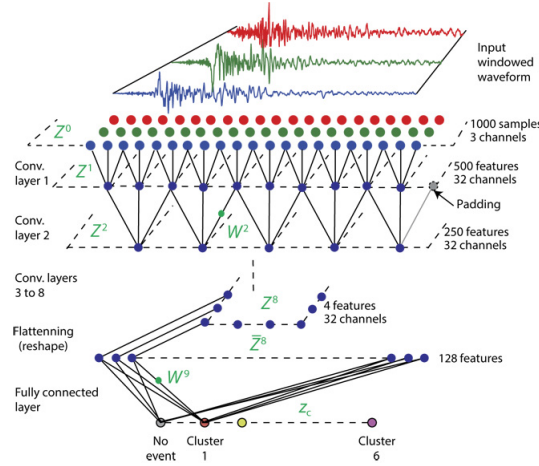
(A) Continuous time series data. (B) Spectrogram: amplitude on log scale. (C) Spectral images from two similar earthquakes at 1267 and 1629 s. (D) Haar wavelet coefficients: amplitude on log scale. (E) Sign of top standardized Haar coefficients after data compression. (F) Binary fingerprint: output of feature extraction. Notice that similar spectral images result in similar fingerprints.

The authors used a continuous data set containing unquoted earthquakes that

probably have similar waveforms. These events are part of the Calaveras fault in central California [40].

### 2.6.4 ConvNetQuake

A highly scalable convolutional neuronal network is presented for the detection and location of earthquakes from a single waveform. It is used to study seismicity induced in Oklahoma, United States, this due to the correlation with the intensification of wastewater injection [18].



**Figure 2.10:** ConvNetQuake Architecture [32]

ConvNetQuake takes a data window of a three-channel waveform seismogram as input and predicts a label; this can be seismic noise or an event and its geographic group (see Figure 2.10). The parameters of the network are optimized to minimize the discrepancy between the predicted labels and the accurate labels in the training set [32].

This model achieves excellent performance in the detection and location of probabilistic events using a single signal; it surpasses other detection methods in computational execution time. The limitation of the methodology is the size of the training set required for a good performance in the detection and location of earthquakes. The increase in

data allowed an excellent performance for the detection of earthquakes [32].

---

## Research Methodology

---

In order to derive the maximum benefit from the research, as well as to communicate accurately the parameters under which it was carried out, it is necessary to establish clearly what was expected to be demonstrated and how to measure the generated contribution to science and the community in general so, in this section we present the Hypothesis that was previously approved by the reviewer committee.

In addition to the above, it was considered that the topic that the project developed was vast, and includes many considerations that could make the project virtually endless. Therefore, together with the objectives, the scope and limitations of this project are presented below, all of which the reviewer committee also approved during the proposal phase of this project.

## 3.1 Hypothesis

Taking into account the problems to be solved and the definitions already explained in this document, it was proposed that:

The use of convolutional neural networks, applied to **time series of seismic events** allows to identify the primary waves with a **F1 score of at least 0.896**, and secondary waves with a **F1 score of at least 0.801** of a seismic event.<sup>a</sup>

$H_0$ :  $conv_{f1} < goal_{f1}$

$conv_{f1}$ : F1 score for our model

$H_1$ :  $conv_{f1} \geq goal_{f1}$

**Goal f1**: goal F1 score based on PhaseNet

---

<sup>a</sup>F1 Current PhaseNet score [52]

This allows that the incorporation of different types of convolutional neural network architectures in time series that justify a higher precision may be taken into account in future efforts to detect, locate, classify and characterize seismic events.

## 3.2 Objectives

### 3.2.1 Main Objective

Identify the time of arrival of the primary and secondary waves of a seismic event, through the use of convolutional neural networks in time series.

### 3.2.2 Specific Objectives

1. Develop Convolutional Neural Network (*ConvNet*) models using different combinations of hyper-parameters, in conjunction with the OVSICORI data to identify the arrival of seismic waves  $P$  and  $S$ .
2. Obtain the F1 score of the models from the experiments described later.
3. Obtain statistical evidence to verify the hypothesis.

## 3.3 Scope and Limitations

Within the scope of this investigation was defined and approved the following set of deliverable:

- Models of convolutional neural networks to identify seismic waves  $P$  y  $S$
- Auxiliary programs for the execution and control of the previous deliverable.
- F1 Score metric of the tested models.
- Statistical analysis to compare the results of the experiments.

It is necessary to delimit this investigation for reasons of time and extension. The following limitations are then raised; it will not be taken into account:

### 3.3. Scope and Limitations

---

- Any algorithm not mentioned in this document.
- The training time for the algorithms is not going to be considered within the metrics.
- Other sample formats, or samples not collected during the investigation to be performed.
- Considerations or implementation of the algorithm for commercial use.
- Any other result, document, software or production that is not included in the deliverable.



---

## Preprocessing Seismic Wave Data

---

In this chapter, we explore how to pre-process the seismic wave time series to create Convolutional Neural Networks that treat the localization of *P-waves* and *S-waves* as a classification problem by using Sliding Windows and a stitching algorithm. We also evaluated different windows sizes and other pre-processing techniques, such as different scaling algorithms, to determine which combination gives the best results for our Neural Networks.

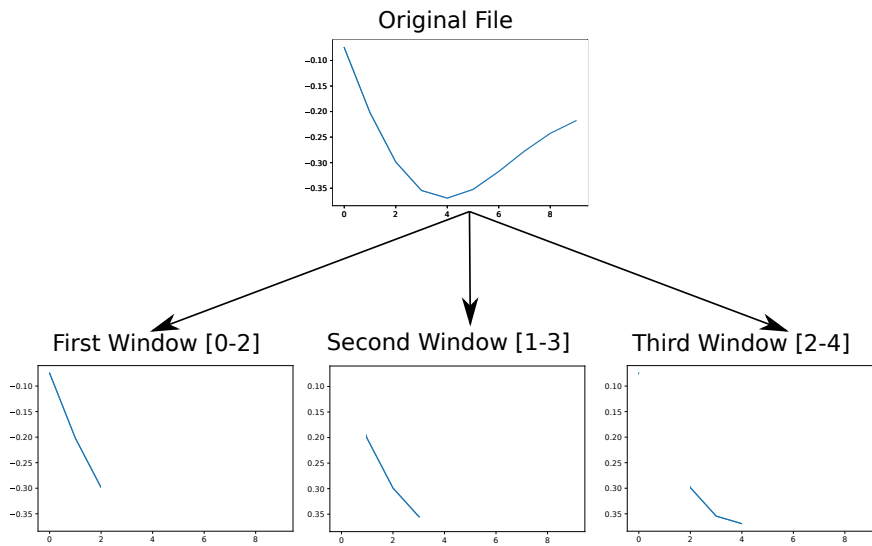
### 4.1 Sliding Windows

To be able to turn this problem which uses seismic wave data, which are sequential time series, into a classification problem we can apply the sliding windows algorithm, which converts the time series into multiple windows which can be classified [12], in our experiments we want to predict the location of the *P-wave* or *S-wave* within the time series, so each window will be classified into one of two labels, the first class *P-wave* or *S-wave* means that the current window that was used as input contains the *P-wave* or *S-wave*, the second class *not-P-wave* or *not-S-wave* means that the window used as input does not contain the *P-wave* or *S-wave*.

The algorithm that we developed takes as input the complete time series (in most of the cases 60 seconds of a seismogram) and separates it into multiple windows of a specified size, when creating these multiple windows we move the window by one element at each step, for instance if we have a time series with 100 observations and use a windows size of 50 observations we will create 100 windows of size 50. The figure on 4.1 describes how we split a single time series into multiple windows.

This way we are able to augment the data, instead of moving the window by 50 each time and having just two windows as input, we create multiple windows that contains

similar but not identical data and create a much bigger data set to train our models.



**Figure 4.1:** Sliding Window example

## 4.2 Scaling

When the data has very different scales and contain some huge outliers, difficulties are created to visualize the data and, what is more important, they can degrade the predictive performance of many machine learning algorithms [31]. Unscaled data can also slow down or even prevent the convergence of many gradient-based estimators. That is why it is recommended to apply some form of normalization, in this case, Scaler is within the ways of normalization mentioned above.

- *StandardScaler*: Removes the mean and scales the data to unit variance. However, the outliers have an influence when computing the empirical mean and standard deviation which shrinks the range of the feature values [31].
- *MinMaxScaler*: Rescales the data set such that all feature values are in the range  $(0, 1)$ [31].

- *Normalizer*: Rescales the vector for each sample to have unit norm, independently of the distribution of the samples [31].

## 4.3 Balancing

The data that we are using is highly unbalanced, this means that there is a high difference between the classes in the data, in this case the count of samples that have a *P-wave* or *S-wave* is much lower than the samples that do not contain *P-wave* or *S-wave*, this is because the *P-wave* or *S-wave* is just a single point in the whole time series, when we do the windows we create at least  $N$  samples of  $P$  where  $N$  is the size of the window, however, this is not enough to balance the data [26].

We decided to use a simple under sampling algorithm to create a balanced data set. Our algorithm consists in selecting fewer samples from one of the classes to predict, in our case we decided to randomly select the same number of samples as the class that have fewer samples. After under sampling our data we end up with the same number of samples that contains the *P-wave* or *S-wave* and samples that does not contain the *P-wave* or *S-wave*.

This helps our models to avoid overfitting due to the high difference of samples for each class. In this case the overfitting comes from the model learning that selecting the most frequent class creates a very high accuracy.

## 4.4 Pre-processing

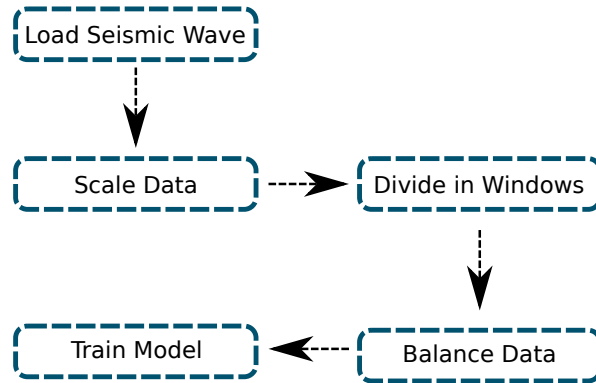
After understanding the process of creating the windows and normalizing the data, we can now describe the complete pre-processing pipeline.

First we start by scaling the data, which consist in using a Scaler algorithm so that all of our data is in the same range of values, normalizing the data we create all the windows of the specified size that we need from the time series.

At the end of this process we have enough data of a fixed size to be used to train and classify the data in a classification algorithm, which in this case are several Convolutional Neural Networks.

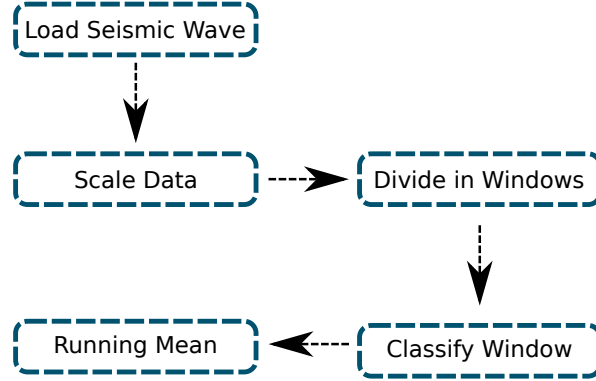
When training the data the extra pre-processing step is the under sampling of data. Once we do the data balancing our data set consist of 50% samples with a *P-wave* or *S-wave* class and of 50% samples with a *not-P-wave* or *not-S-wave* class.

The figure 4.2 shows a high level view of what our process looks like.



**Figure 4.2:** Pre-processing and training flow

Our models takes as input an array of window and as output a probability of that window to contain the *P-wave* or *S-wave*. After we have all of the probabilities of each window containing the *P-wave* or *S-wave* we must now calculate the probabilities for the original time series where the *P-wave* or *S-wave* to determine where the *P-wave* or *S-wave* is.



**Figure 4.3:** Pre-processing and prediction flow

We do this by stitching and using a convolution to calculate the running mean of the results and getting back an array of the same size of the original time series, each having the probability of the *P-wave* or *S-wave* being at that index. The prediction process can be visualized in figure 4.3.

---

## Experimentation

---

## 5.1 Experimentation Setting

To be able to verify if there is evidence supporting our hypotheses, we proposed doing a set of experiments by using Design of Experiments and analysing the results using ANOVA. The experiments consisted in testing several factors to see their effect in the f1 score, these factors included:

- Architecture Factor: The type of neural network (classification model) to use.
- Layer: number of layers for the neural network model.
- Window size: the size of the window to use as input to the classification model.
- Scaler: the algorithm to scale the data.

To be able to understand the experiments at best, we will first describe the data set that was used to train and validate our models.

### 5.1.1 Data Set

In order to be able to train and validate the classification models, we are going to use the information compiled by the **Northern California Earthquake Data Center - UC Berkeley Seismic Laboratory** [27].

#### 5.1.1.1 Training

The data set that we used for  $P$  seismic wave contains data for one network, 32 stations and three channels. The total of unique events was of **13 985**, in this events there were **118 092** samples of the  $P$ -wave.

The data set that we used for  $S$  seismic wave contains data for one network, 32 stations and six channels. The total of unique events was of **13 253**, in this events



there were **74 026** samples of the *S-wave*.

This data was collected by the **Northern California Earthquake Data Center - UC Berkeley Seismic Laboratory** [27] and the data set contains samples from January 1st, 2014 to March 31th, 2014.

### 5.1.1.2 Validation

Similarly, data validation was against totally different data sets, the validation data comes from 1 network, 33 stations and three channels. A total of **2 012** events for *P* waves. In this events there were **16 720** samples of the *P-wave*.

For *S* waves the data was collected from one network, 33 stations and seven channels for a total of **1 936** events for *S* waves. In this events there were **11 368** samples of the *S-wave*.

This data was also collected by the **Northern California Earthquake Data Center - UC Berkeley Seismic Laboratory** [27] and the data set contains samples from April 1st, 2014 to April 30th, 2014.

It is very important to understand that the validation data comes from a different set of dates than the training data which guarantees that the models were validated with a completely different data set.

### 5.1.2 Pre-Experiments

Before we tried to test our hypotheses we wanted to gather enough information to create better models, that's why we decided to perform pre experiments which would tell us which models performed best and then test our hypotheses using those models.

First we started by experimenting with Convolutional Neural Networks.

### 5.1.2.1 Convolutional Neural Network Model

We did an experiment to find the best ConvNet architecture. For this pre-experiment we used the number of layers as a factor with three levels (see table 5.1). For this pre-experiment we did 102 repetitions. The total number of executions was,  $3 \times 102 = 306$ .

Net	Layer
Conv	1
	3
	5

**Table 5.1:** Table with Factors and Levels of Pre-Experiments for Conv Net

#### 5.1.2.1.1 ConvNet Implementation

The listings 5.1 shows the code used to create the ConvNet models used in this experiment. A summary of the Convolutional Model with five layers is in the listing 5.2.

```

1 def network(input_window_size=60, filter_number=1, conv_window=(3,), pooling_window=(2,),
2             dropout_rate=[0.3], activation="relu", dense_activation="softmax", optimizer=("adam",0),
3             loss="categorical_crossentropy", layers=1, l1_value=0.0001, l2_value=0.0001):
4     model = Sequential()
5
6     # Input Layer
7     model.add(Conv1D(filter_number, conv_window, activation=activation, padding="same",
8                     input_shape=(input_window_size, 1),
9                     activity_regularizer=l1_l2(l1=l1_value, l2=l2_value)))
10    model.add(MaxPooling1D(pooling_window, padding="same"))
11    model.add(Dropout(dropout_rate[0]))

```

## 5.1. Experimentation Setting

---

```
12
13     # Hidden Layers
14     current_filter = 1
15     filter_number_temp = filter_number
16     for i in range(layers):
17         filter_number_temp = filter_number_temp * 2
18         model.add(Conv1D(filter_number_temp, conv_window, activation=activation, padding="same",
19                         activity_regularizer=l1_l2(l1=l1_value, l2=l2_value)))
20         model.add(MaxPooling1D(pooling_window, padding="same"))
21         model.add(Dropout(dropout_rate[current_filter]))
22         current_filter = current_filter + 1
23
24     # Output Layer
25     model.add(Flatten())
26     model.add(Dense(2, activation=dense_activation))
27
28     model.compile(optimizer=optimizer[0], loss=loss, metrics=["categorical_accuracy"])
29
30     return model
```

**Listing 5.1:** Convolutional Neural Network with Layer Parameter Implementation

1	-----		
2	Layer (type)	Output Shape	Param #
3	=====		
4	conv1d_6 (Conv1D)	(None, 60, 1)	4
5	-----		
6	max_pooling1d_6 (MaxPooling1	(None, 30, 1)	0
7	-----		
8	dropout_5 (Dropout)	(None, 30, 1)	0
9	-----		
10	conv1d_7 (Conv1D)	(None, 30, 2)	8
11	-----		
12	max_pooling1d_7 (MaxPooling1	(None, 15, 2)	0

## 5.1. Experimentation Setting

---

13	-----		
14	dropout_6 (Dropout)	(None, 15, 2)	0
15	-----		
16	conv1d_8 (Conv1D)	(None, 15, 4)	28
17	-----		
18	max_pooling1d_8 (MaxPooling1	(None, 8, 4)	0
19	-----		
20	dropout_7 (Dropout)	(None, 8, 4)	0
21	-----		
22	conv1d_9 (Conv1D)	(None, 8, 8)	104
23	-----		
24	max_pooling1d_9 (MaxPooling1	(None, 4, 8)	0
25	-----		
26	dropout_8 (Dropout)	(None, 4, 8)	0
27	-----		
28	conv1d_10 (Conv1D)	(None, 4, 16)	400
29	-----		
30	max_pooling1d_10 (MaxPooling	(None, 2, 16)	0
31	-----		
32	dropout_9 (Dropout)	(None, 2, 16)	0
33	-----		
34	conv1d_11 (Conv1D)	(None, 2, 32)	1568
35	-----		
36	max_pooling1d_11 (MaxPooling	(None, 1, 32)	0
37	-----		
38	dropout_10 (Dropout)	(None, 1, 32)	0
39	-----		
40	flatten_2 (Flatten)	(None, 32)	0
41	-----		
42	dense_2 (Dense)	(None, 2)	66
43	=====		
44	Total params: 2,178		
45	Trainable params: 2,178		

46 `Non-trainable params: 0`

47 -----

**Listing 5.2:** Convolutional Neural Network with Layer Parameter Summary

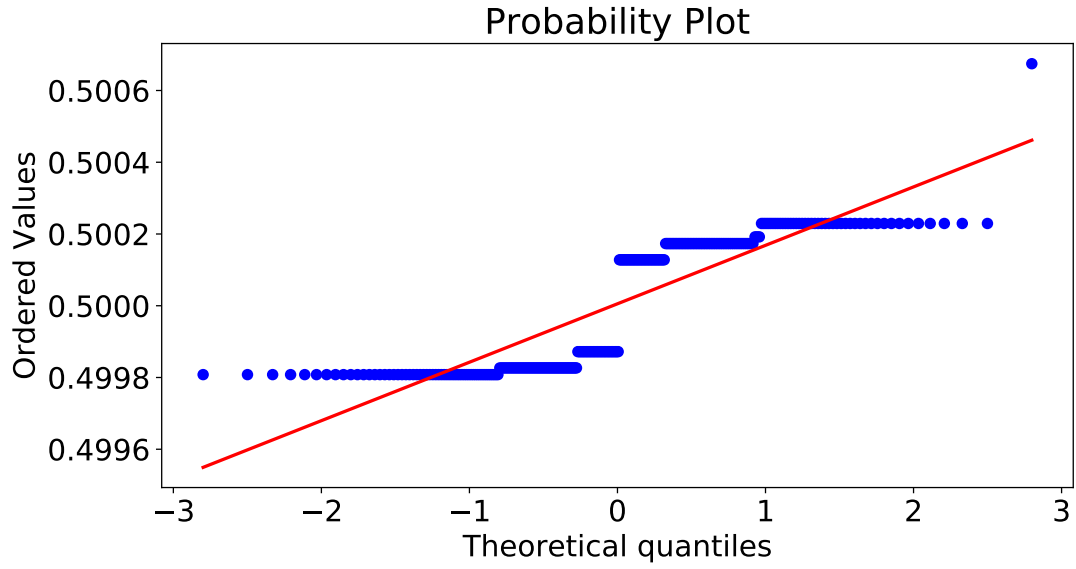
### 5.1.2.1.2 ConvNet Experiments Results

These models didn't perform very well because the models demonstrated that they weren't able to learn, the output of the experiment showed an accuracy near the 50% (see table 5.2), which is as good as random when classifying between two classes.

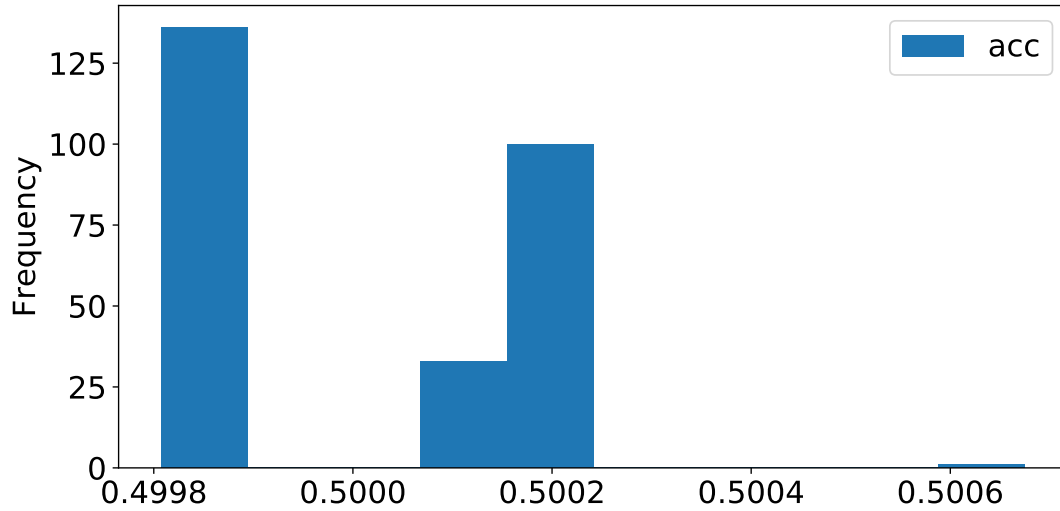
Net	Layer	Average Accuracy
Conv	1	0.499949478
Conv	3	0.5000151645
Conv	5	0.4999819415

**Table 5.2:** Table with Average Accuracy Results for Pre-Experiments

Before discarding this models and the data we decided that a statistical test was needed in order to confirm that all of those models were almost random classification, we decided to use the non-parametric Kruskal-Wallis test as the data is not normal and didn't meet ANOVA assumptions see figures 5.1 and 5.2.



**Figure 5.1:** Pre-Experiment Convolutional - Normal Q-Q plot.



**Figure 5.2:** Pre-Experiment Convolutional - Histogram plot

The Kruskal-Wallis test allows us to compare several samples to test testing whether or not these samples originate from the same distribution [9].

After checking the results, all of the models performed about the same (there was

no difference between them) and the accuracy was around 50%.

Based on these results we decided that these models should not be used in our final experiment and created another Convolutional Neural Network Model based on a Le-Net model.

### 5.1.2.1.3 Le-Net based ConvNet

For our new implementation we decided to base our model on a Le-Net architecture. The first two layers consist on a *Convolutional* layer followed by a *Max Pooling* layer and then it flattens out, then we used a *Dense* layer followed by a second *Dense* layer with an output size of the number of classes that our model must classify.

The code listing 5.3 shows the architecture for the convolutional model previously explained and the code listing 5.4 shows a high level view of the architecture.

```
1 def conv_net_2(input_window_size=60, filter_number=32, conv_window=(3,), pooling_window=(2,),
2               dropout_rate=[], activation="relu", dense_activation="softmax", optimizer="adam",
3               loss="categorical_crossentropy", layers=1, l1_value=0.0001, l2_value=0.0001):
4
5     model = Sequential()
6     model.add(Conv1D(32, (3, ), activation="relu", input_shape=(input_window_size, 1)))
7     model.add(MaxPooling1D((2,)))
8     model.add(Conv1D(64, (3,), activation="relu"))
9     model.add(MaxPooling1D((2,)))
10    model.add(Conv1D(64, (3,), activation="relu"))
11    model.add(Conv1D(64, (3,), activation="relu"))
12    model.add(Flatten())
13    model.add(Dense(128, activation="relu"))
```

## 5.1. Experimentation Setting

---

```
14 model.add(Dense(2, activation="softmax"))
15 model.compile(optimizer="adam",
16               loss="categorical_crossentropy",
17               metrics=["categorical_accuracy"])
18 return model
```

**Listing 5.3:** Convolutional Neural Network Two Implementation

```
1
2 Layer (type)                Output Shape          Param #
3 =====
4 conv1d (Conv1D)             (None, 58, 32)        128
5 -----
6 max_pooling1d (MaxPooling1D) (None, 29, 32)        0
7 -----
8 conv1d_1 (Conv1D)           (None, 27, 64)        6208
9 -----
10 max_pooling1d_1 (MaxPooling1 (None, 13, 64)        0
11 -----
12 conv1d_2 (Conv1D)           (None, 11, 64)        12352
13 -----
14 conv1d_3 (Conv1D)           (None, 9, 64)         12352
15 -----
16 flatten (Flatten)           (None, 576)           0
17 -----
18 dense (Dense)               (None, 128)           73856
19 -----
20 dense_1 (Dense)             (None, 2)             258
21 =====
22 Total params: 105,154
23 Trainable params: 105,154
24 Non-trainable params: 0
```



**Listing 5.4:** Convolutional Neural Network Two Summary**5.1.2.2 Dense Network Model**

We did another experiment to find the best Dense architecture. For this experiment we used the number of layers as a factor, in this case we used three levels (see table 5.3). For this experiment we did 60 repetitions. The total for experiments was  $3 \times 60 = 180$ .

The idea is to find the best Dense architecture to compare with the Convolutional Neural Network Model.

Net	Layer
Dense	1
	3
	5

**Table 5.3:** Table with Factors and Levels of Experiments of  $P$  Seismic Wave for Dense Net

**5.1.2.2.1 Dense Model Implementation**

The listings 5.5 shows the code used to create the Dense models used in this experiment. A summary of the Dense Model with five layers is in the listing 5.6

```
1 def dense_net(input_window_size=60, dropout_rate=[], activation="relu",
2               dense_activation="softmax", optimizer="adam",
3               loss="categorical_crossentropy", layers=1, l1_value=0.0001,
4               l2_value=0.0001):
5
6     model = Sequential()
7
```

## 5.1. Experimentation Setting

---

```
8     hidden_units = 120
9     dropout_index = 0
10    model.add(Dense(hidden_units, input_dim=input_window_size, activation=activation))
11    model.add(Dropout(dropout_rate[dropout_index]))
12    dropout_index = dropout_index + 1
13
14    for i in range(layers):
15        model.add(Dense(hidden_units * (i + 2),
16                        input_dim=input_window_size, activation=activation))
17        model.add(Dropout(dropout_rate[dropout_index]))
18        dropout_index = dropout_index + 1
19
20    model.add(Dense(2, activation=dense_activation))
21
22    model.compile(optimizer=optimizer, loss=loss, metrics=["categorical_accuracy"])
23    return model
```

**Listing 5.5:** Dense Network Implementation

```
1
2 -----
3 Layer (type)                Output Shape          Param #
4 =====
5 dense_29 (Dense)             (None, 120)           7320
6 -----
7 dropout_23 (Dropout)         (None, 120)           0
8 -----
9 dense_30 (Dense)             (None, 240)           29040
10 -----
11 dropout_24 (Dropout)         (None, 240)           0
12 -----
13 dense_31 (Dense)             (None, 360)           86760
14 -----
15 dropout_25 (Dropout)         (None, 360)           0
```

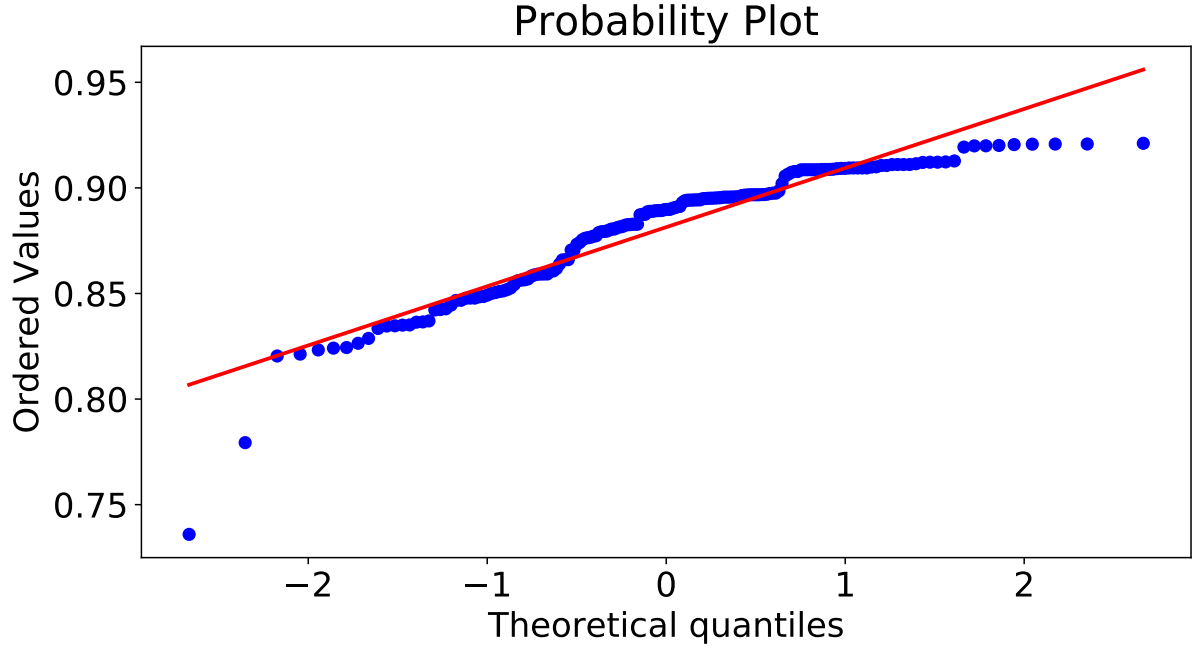
## 5.1. Experimentation Setting

---

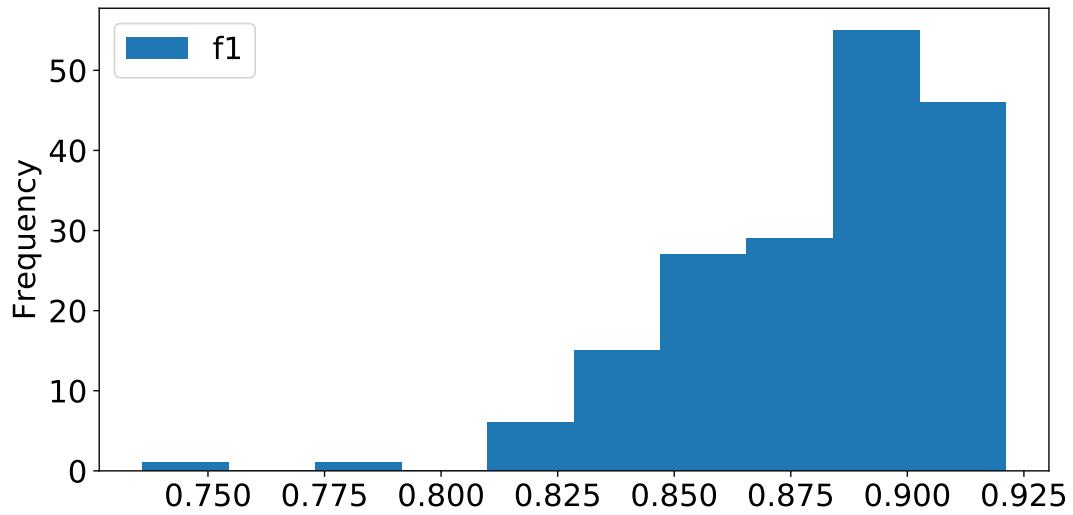
```
16 -----
17 dense_32 (Dense)          (None, 480)          173280
18 -----
19 dropout_26 (Dropout)      (None, 480)           0
20 -----
21 dense_33 (Dense)          (None, 600)          288600
22 -----
23 dropout_27 (Dropout)      (None, 600)           0
24 -----
25 dense_34 (Dense)          (None, 720)          432720
26 -----
27 dropout_28 (Dropout)      (None, 720)           0
28 -----
29 dense_35 (Dense)          (None, 2)             1442
30 =====
31 Total params: 1,019,162
32 Trainable params: 1,019,162
33 Non-trainable params: 0
34 -----
```

**Listing 5.6:** Dense Network Summary

**5.1.2.2.2 Dense Experiments Results** For these experiments the results, again showed that the data was not normal and did not meet ANOVA assumptions, see figures 5.3 and 5.4. However, we decided to use Kruskal-Wallis again in order to analyze our results.

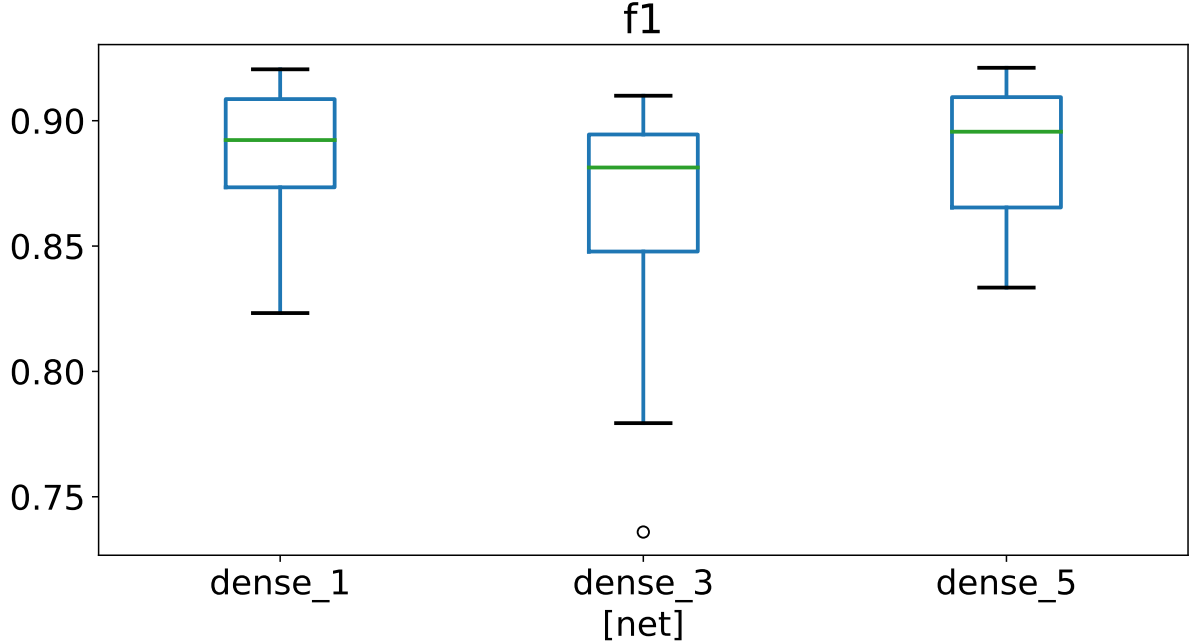


**Figure 5.3:** Pre-Experiment Dense - Normal Q-Q plot.



**Figure 5.4:** Pre-Experiment Dense - Histogram plot

This time the Kruskal-Wallis test showed us that there was a statistical difference between the number of layers used in the experiments, the Dense network with five layers is the one that performs best according to figure 5.5.



**Figure 5.5:** Graph showing the difference of accuracy per layer levels in Dense model

### 5.1.3 Final Experiments

The final experiments are the ones that will let us validate our hypothesis, these consisted of comparing the performance, by using the F1 score, of the Convolutional Neural Network model that we are going to call **conv\_2** (Le-Net implementation). Table 5.4 shows the factors and levels for  $P$  Seismic Wave experiments and  $S$  Seismic Wave experiments.

For the experiments repetitions, we did five repetitions for the  $P$  Seismic Wave and seven repetitions for the  $S$  seismic wave. These seismic waves have different experiments and analyses because we can split into two sub-hypothesis, the central hypothesis. Also, additional of the hypothesis, we decided to compare the performance of Dense and Convolutional Neural Networks (**dense\_5**).

Net	Scaler	Window
Conv_2	None	30
Dense_5	Standard Scaler	60

**Table 5.4:** Table with Factors and Levels of Experiments

- Net: The different architecture of neural network.
- Scaler: This factor define the scaling algorithm should be use 4.2.
- Window size: This defines the window size to use sliding windows 4.1.

Using the factors and levels on table 5.4 we did a total of  $2 * 2 * 2 = 8$  experiments, which were repeated 5 times, for a total  $8 * 5 = 40$  executions for P Seismic Wave. Also using the factors and levels on table 5.4 we did a total of  $2 * 2 * 2 = 8$  experiments, which were repeated 7 times, for a total  $8 * 7 = 56$  executions.

As part of the experiments we need to use our preprocessing strategy described in the chapter 4. The implementation of this pre-processing algorithm is described in the next sub section.

#### 5.1.4 Preprocessing Seismic Wave Data Implementation

To be able to use the seismic wave data in our neural network models we need to pre-process the data, we use the scaler and window size factor to process the data and then use our models to classify each window.

The first phase of this is to download the data, the listing 5.7 shows the coded used to download the data set. After the download process is completed we generate files (.h5 raw) with the function in the listing 5.8, this files are ready to be pre-processed

by our algorithm.

```

1 def download_data(start, end, verbose=True):
2     output_file = output_catalog_folder.format(start, end)
3     request_url = catalog_url.format(start=start, end=end)
4
5     return output_file
6
7 def get_waveform(network, station, channel, start_time, end_time, verbose=True):
8     output_file = output_waveform_folder.format(network, station, channel, start_time, end_time)
9     request_url = waveform_url.format(network=network, station=station, channel=channel,
10                                     start_time=start_time, end_time=end_time)
11     return output_file
12
13 def generate_dataset(output_filename, start, end, verbose=True):
14     downloaded_path = download_data(start, end, verbose)
15     catalog = read_events(downloaded_path)
16     number_events = len(catalog)
17     with open(output_filename, "w+") as data:
18         data.write("network,station,channel,time,phase,waveform,"
19                 "event,event_type,magnitude,magnitude_type\n")
20     for index, event in enumerate(catalog):
21         # Check for 10 arrivals per event to maximize the amount
22         # of events while still having 10 samples per event
23         p_arrivals = 0
24         s_arrivals = 0
25         for arrival, pick in zip(event.origins[0].arrivals, event.picks):
26             if arrival.phase == "P" and p_arrivals < 10:
27                 network = pick.waveform_id.network_code
28                 station = pick.waveform_id.station_code
29                 channel = pick.waveform_id.channel_code
30                 start_time = (pick.time - timedelta(seconds=10))

```

```

31         .strftime("%Y-%m-%dT%H:%M:%S")
32     end_time = (pick.time + timedelta(seconds=20))
33         .strftime("%Y-%m-%dT%H:%M:%S")
34     waveform = get_waveform(network, station, channel,
35                             start_time, end_time, verbose)
36     if os.stat(waveform).st_size > 0:
37         data.write("{} , {} , {} , {} , {} , {} , {} , {} , {} , {} \n"
38                 .format(network, station, channel, pick.time, arrival.phase,
39                         waveform, event.resource_id, event.event_type,
40                         event.magnitudes[0].mag, event.magnitudes[0].magnitude_type))
41         p_arrivals = p_arrivals + 1
42         os.remove(waveform)
43     elif arrival.phase == "S" and s_arrivals < 10:
44         network = pick.waveform_id.network_code
45         station = pick.waveform_id.station_code
46         channel = pick.waveform_id.channel_code
47         start_time = (pick.time - timedelta(seconds=10))
48             .strftime("%Y-%m-%dT%H:%M:%S")
49         end_time = (pick.time + timedelta(seconds=20))
50             .strftime("%Y-%m-%dT%H:%M:%S")
51         waveform = get_waveform(network, station, channel,
52                                 start_time, end_time, verbose)
53         if os.stat(waveform).st_size > 0:
54             data.write("{} , {} , {} , {} , {} , {} , {} , {} , {} , {} \n"
55                     .format(network, station, channel, pick.time, arrival.phase,
56                             waveform, event.resource_id, event.event_type,
57                             event.magnitudes[0].mag, event.magnitudes[0].magnitude_type))
58             s_arrivals = s_arrivals + 1
59             os.remove(waveform)
60         if s_arrivals >= 10 and p_arrivals >= 10:
61             break
62     time.sleep(0.25)
63

```



```
64     return pd.read_csv(output_filename)
```

**Listing 5.7:** Download Data from Northern California Earthquake DataCenter

```
1 def generate_h5_files(start_date, end_date, output_path, phase="P", dataframe_file=None):
2     if dataframe_file is None:
3         dataframe_file = tempfile.NamedTemporaryFile().name
4     # Get data from webservice
5     dataset = generate_dataset(dataframe_file, start_date, end_date)
6     with open("failures.csv", "a") as failures:
7         for index, row in dataset.iterrows():
8             # Load waveform
9             tr = read(row["waveform"])[0]
10            tr.filter("bandpass", freqmin=1.0, freqmax=10.0, corners=4, zerophase=True)
11            tr.normalize()
12            # Get pick location
13            pick_time = UTCDateTime(row["time"])
14            pick_index = int((pick_time - tr.stats.starttime) * tr.stats.sampling_rate)
15            # Data to save to h5
16            X = tr.data
17            Y = np.zeros(len(X))
18            Y[pick_index] = 1
19            # Save h5 file
20            base = os.path.basename(row["waveform"])
21            new_file_path = os.path.join(output_path, "{}.h5"
22                .format(os.path.splitext(base)[0]))
23            h5f = h5py.File(new_file_path, "w")
24            h5f.create_dataset("normal", data=X)
25            h5f.create_dataset("transformed", data=Y)
26            h5f.close()
```

**Listing 5.8:** Generation Files

The script used to pre-process the seismic wave data can be found on the listing 5.9.

```

1 def rolling_window(a, window, step_size, padding=True, copy=False):
2     if copy:
3         result = a.copy()
4     else:
5         result = a
6     if padding:
7         result = np.hstack((result, np.zeros(window)))
8     shape = result.shape[:-1] + (result.shape[-1] - window + 1 - step_size, window)
9     strides = result.strides + (result.strides[-1] * step_size,)
10    return np.lib.stride_tricks.as_strided(result, shape=shape, strides=strides)
11
12 def load_data(size, base_path, scaler="none", normalizer=False):
13     total_files = len(os.listdir(base_path))
14     x_dataset = np.zeros(shape=(0, size))
15     y_dataset = np.zeros(shape=(0, 1))
16     for file_path in os.listdir(base_path):
17
18         # Load data
19         h5f = h5py.File(os.path.join(base_path, file_path), "r")
20         x = h5f["normal"][:]
21         y = rolling_window(h5f["transformed"][:], size, 1)
22
23         # Scale data before create windows
24         if scaler != "none":
25             if scaler == "standard_scaler":
26                 scaler = StandardScaler()
27                 scaler = scaler.fit(x.reshape(-1, 1))
28                 x = scaler.transform(x.reshape(1, -1))[0]
29
30         x = rolling_window(x, size, 1)
31
32         # Determine if the wave is in the window or not
33         y = [[np.amax(array)] for array in y]

```

```
34     x_dataset = np.vstack((x_dataset, x))
35     y_dataset = np.vstack((y_dataset, y))
36
37     x_dataset = np.reshape(x_dataset, (len(x_dataset), size, 1))
38     y_dataset = np.reshape(y_dataset, (len(y_dataset), 1))
39     # Balancing
40     # n = size of which there are less labels
41     n = min(len(np.where(y_dataset == 1)[0]), len(np.where(y_dataset == 0)[0]))
42
43     # Choose n random samples from each label
44     mask = np.hstack([np.random.choice(np.where(y_dataset == 1)[0], n, replace=False)
45                        for l in np.unique(y_dataset)])
46     x_dataset = x_dataset[mask]
47     y_dataset = y_dataset[mask]
48
49     # create a one-hot-encoding vector of y_dataset
50     y_dataset = to_categorical(y_dataset)
51
52     return x_dataset, y_dataset
```

**Listing 5.9:** Preprocessing Data

First we load the file and divide into windows of the size given by the *Window* level, then if the `standard_scaler` is present we scale the data, next we determine if the wave ( $P$  or  $S$ ) is in the window or not to balancing the data and finally we have the data pre-processed.

## 5.2 Experimentation Execution

As a first step to execute the experiments, all the possible combinations of the experiments were created and then randomized in a file to guarantee independence for ANOVA, we then run the experiments as formalized in the listing 5.10 in the randomized

order defined in the file created.

```
1 def run_experiment(experiment, settings):
2     net_name = experiment[0]
3     window_size = experiment[1]
4     scaler = experiment[2]
5     normalizer = experiment[3]
6
7     input_folder = settings["input_folder"]
8     dropout_rates = settings["dropout_rates"]
9     batch_size = settings["batch_size"]
10    epochs = settings["epochs"]
11
12    network = network_map[net_name](input_window_size=window_size,
13                                    dropout_rate=dropout_rates)
14
15    x, y = load_data(input_folder, window_size, scaler, normalizer)
16    x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=4)
17
18    if net_name == "dense_5":
19        x_train = x_train.reshape(len(x_train), window_size)
20        x_test = x_test.reshape(len(x_test), window_size)
21
22    early_stop = EarlyStopping(monitor="val_categorical_accuracy", restore_best_weights=True,
23                                patience=20, min_delta=0.01, verbose=1)
24    history = network.fit(x_train, y_train, batch_size, epochs, validation_data=(x_test, y_test),
25                            callbacks=[early_stop], verbose=2)
26
27    identifier = uuid.uuid4().hex[:6].lower()
28
29    network.save("results/{_}_{_}_{_}_{_}_{_}.h5".
30                format(net_name, window_size, scaler, normalizer,
```

```
31         history.history["val_categorical_accuracy"][-1],
32         history.history["val_loss"][-1], identifier))
33
34 with open("/experiments.json") as json_file:
35     experiments = json.loads(json_file.read())
36
37 for index, experiment in enumerate(experiments["randomized"][start:end]):
38     run_experiment(experiment, experiments["settings"])
```

**Listing 5.10:** Run Experiments

### 5.2.1 Results

Once all the experiments were executed, we had to calculate the F1 score, this was done by the code in the listing 5.11. The results were written in a comma separated value file for further analysis in Python and R, the results are attached in the annexes.

```
1 model_directory = "/results/"
2 file_directory = "/data/"
3
4 def calculate_f1_score(model, x_data_set, y):
5     model_exp = load_model(model)
6     output_predict = np.array([np.argmax(x) for x in model_exp.predict(x_data_set)],
7                               dtype=np.int)
8     recall = recall_score(y, output_predict)
9     precision = precision_score(y, output_predict)
10    F1_score = f1_score(y, output_predict)
11    results = [recall, precision, F1_score]
12    return results
13
14
15 def execute_predict_and_calculation():
```

```
16 with open("/calculate_f1_score.csv", "a") as f1_score:
17     for file_name in os.listdir(file_directory):
18         path = os.path.join(file_directory, file_name)
19         h5f = h5py.File(path, "r")
20         x = h5f["x"][:]
21         y = np.array(h5f["Y"][:], dtype=np.int)
22         y = y.reshape(len(y))
23         h5f.close()
24         params = os.path.splitext(file_name)[0].split("_")
25         params = "*".join(params)+"*"
26         model_path = model_directory+"*"+params+".h5"
27         models = glob.glob(model_path)
28         for model in models:
29             params_net = os.path.splitext(model.split("/")[-1])[0].split("__")
30             if params_net[0] == "dense":
31                 x = x.reshape(len(x), int(params_net[2]))
32             else:
33                 x = x.reshape(len(x), int(params_net[2]), 1)
34             results = calculate_f1_score(model, x, y)
35             f1_score.write(f"{model},{results[0]},{results[1]},{results[2]}\n")
36
37 execute_predict_and_calculation()
```

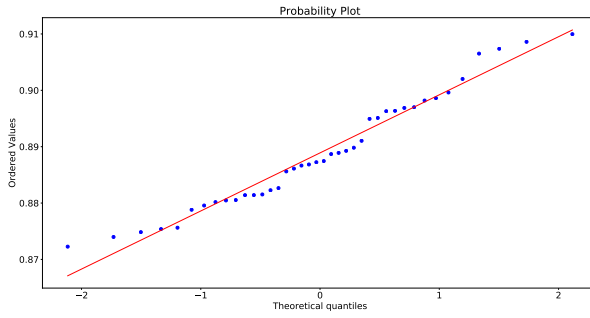
**Listing 5.11:** Calculate F1 Score

Previous to the analysis using ANOVA, we need to make sure that the ANOVA assumptions are met.

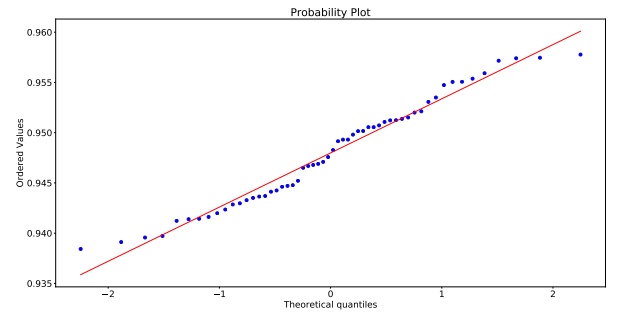
- **Normality:** We meet with this assumption, the figures 5.6, 5.7 show the Normal Q-Q plot and the figures 5.8, 5.9 show the residuals vs. fitted values.
- **Homogeneity of variance:** After performing the Levene-Test (see 5.12) it shows that the data comply with this assumption table 5.5 show the results of the Levene test for both experiments.

- **Independent observations:** The observations gathered in this research were obtained by executing the experiments in random order which makes the data compliant with this assumption.

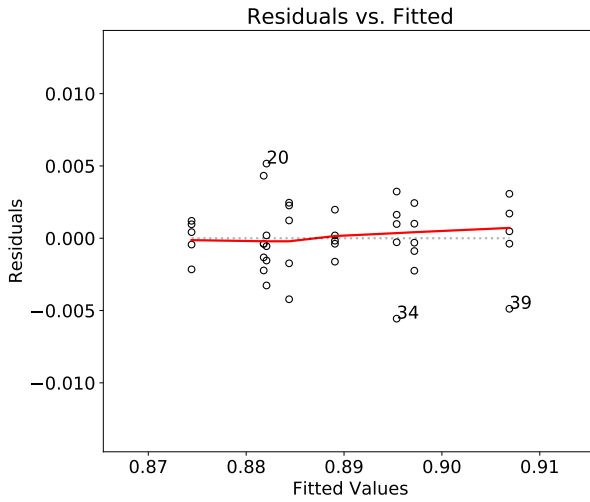
It is also important to note that even if the normality assumption was not met entirely, ANOVA is robust to violations of the normality assumption, especially when the sample size is relatively large, and should not be cause for substantial concern [22].



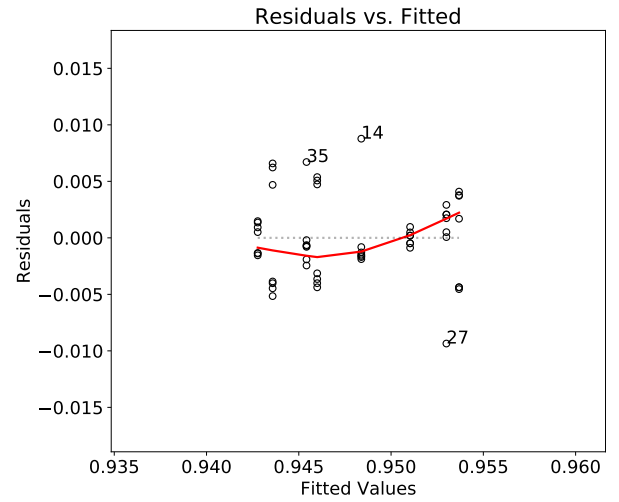
**Figure 5.6:** Normal Q-Q plot –  $P$  wave.



**Figure 5.7:** Normal Q-Q plot –  $S$  wave.



**Figure 5.8:** Residuals vs Fitted –  $P$  wave.



**Figure 5.9:** Residuals vs Fitted –  $S$  wave.

```

1 populations = []
2 for net in p_metrics.reset_index()["net"].unique():
3     for window in p_metrics.reset_index()["window"].unique():
4         for scaler in p_metrics.reset_index()["scaler"].unique():
5             populations.append(p_metrics.reset_index()
6                               .query(f"net == \"{net}\"
7                                     & scaler == \"{scaler}\"
8                                     & window == \"{window}\"")["f1"])
9
10 levene_result = stats.levene(*populations)[1]
```

**Listing 5.12:** Levene Test

Levene Results P wave	Levene Results S wave
p-value: 0.8835	p-value: 0.2586

**Table 5.5:** Table with Levene Results of  $P$  and  $S$  Seismic Wave

After making sure that we are able to use ANOVA to analyze our experiments we got the results in listings 5.13 for *Experiments P Seismic Wave* and 5.14 for *Experiments*



### *S Seismic Wave.*

```

1 Anova Table (Type II tests)
2
3 Response: f1
4
5      Sum Sq Df F value    Pr(>F)
6 net      0.00079377  1 122.2580 1.835e-12 ***
7 window    0.00199414  1 307.1410 < 2.2e-16 ***
8 scaler    0.00093492  1 143.9981 2.193e-13 ***
9 net:window 0.00002078  1   3.2004  0.083091 .
10 net:scaler 0.00000172  1   0.2649  0.610318
11 window:scaler 0.00006190  1   9.5333  0.004148 **
12 net>window:scaler 0.00000051  1   0.0778  0.782034
13 Residuals      0.00020776 32
14 ---
15 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

**Listing 5.13:** ANOVA - Experiments P Seismic Wave

```

1 Anova Table (Type II tests)
2
3 Response: f1
4
5      Sum Sq Df F value    Pr(>F)
6 net      0.00006159  1   4.2730  0.04414 *
7 window    0.00070216  1 48.7151 7.901e-09 ***
8 scaler    0.00000419  1   0.2910  0.59205
9 net:window 0.00000262  1   0.1816  0.67188
10 net:scaler 0.00006546  1   4.5412  0.03824 *
11 window:scaler 0.00000262  1   0.1816  0.67193
12 net>window:scaler 0.00003019  1   2.0948  0.15430
13 Residuals      0.00069186 48
14 ---

```

```
14 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 1
```

### **Listing 5.14:** ANOVA - Experiments S Seismic Wave

The results described in the listings 5.13 and 5.14 show that the difference of the means of `f1_score` caused by the network, window, scaler, and normalizer factors are very significant, and we get that some interactions are significant too, this allows us to analyze the different factors and make conclusions on the means of them.

---

## Discussion

---

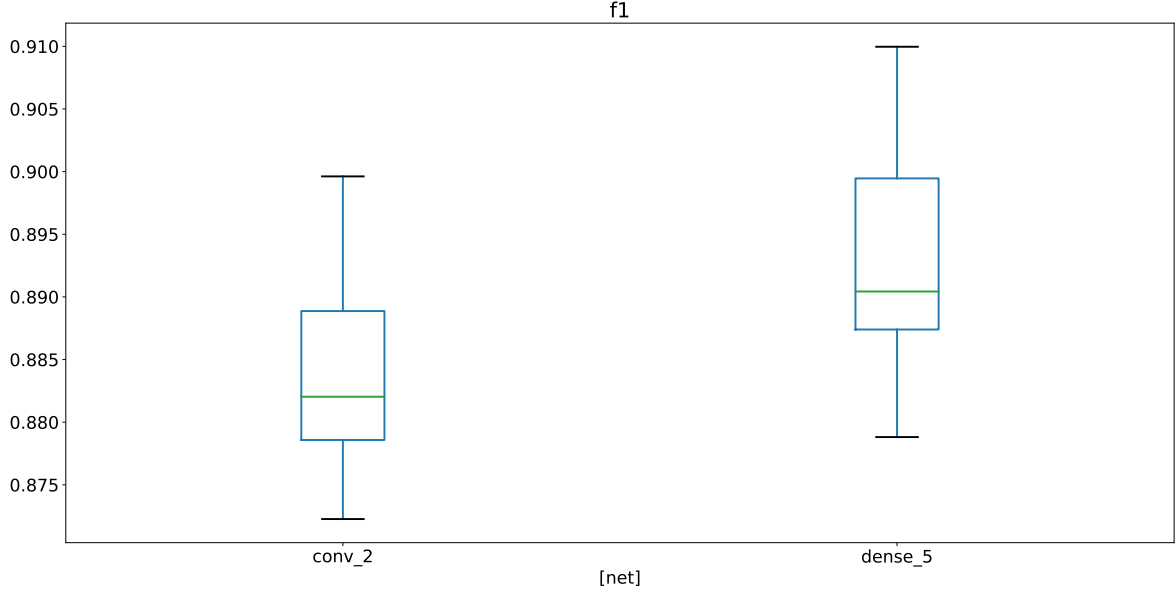
This section contains a detailed discussion of the results of the experiments. The ANOVA results displayed on the listings 5.13 and 5.14 show us that we have some factors and interactions that we can analyze, we will focus on those that show high significance, however we also mention some that are not statistically different.

First we will start by analyzing individual factors for both the *P-wave* and the *S-wave* and then we will move to the interactions.

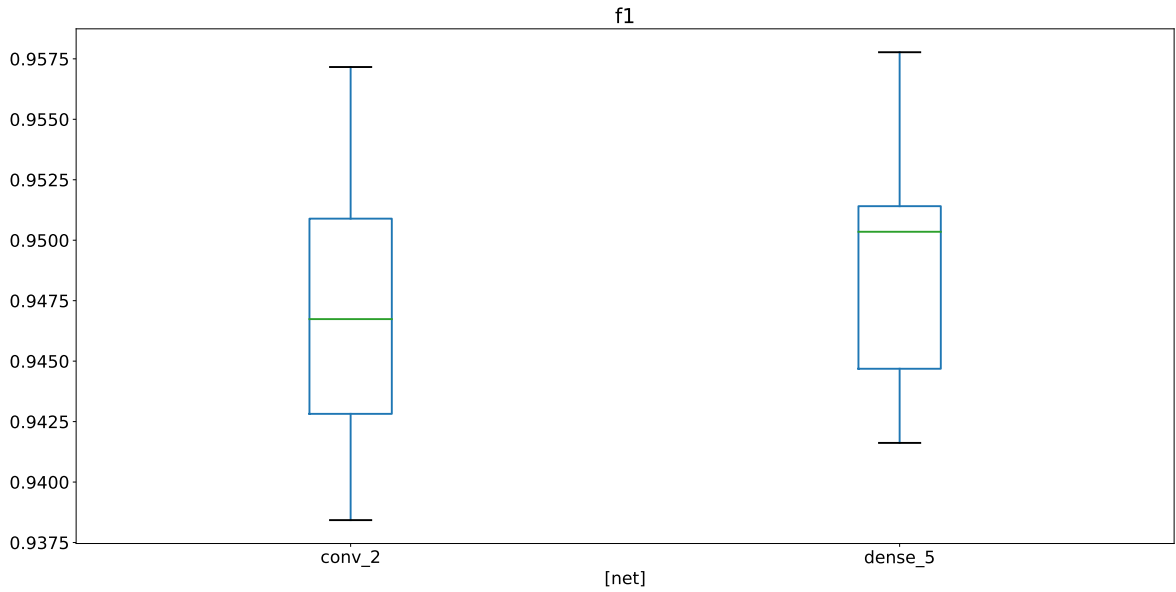
### 6.1 Net

This factor is one of the most important for us since it shows the difference between the Dense Neural Network and the Convolutional Neural Network. In both experiments it is a significant factor,  $< 0.01$  for *P-Wave Experiments* and  $< 0.05$  for *S-Wave Experiments*, from which we can conclude that they are statistically different.

In the figures 6.1 and 6.2 we can see that the *dense\_5* network performs better than the *conv\_2* network, however they perform very similarly and something very interesting here is that even when the *dense\_5* contains around 9 times more trainable parameters than the *conv\_2*, which makes it less computer intensive and able to achieve similar results.



**Figure 6.1:** Graph showing the difference of score per network levels in the *P-wave*

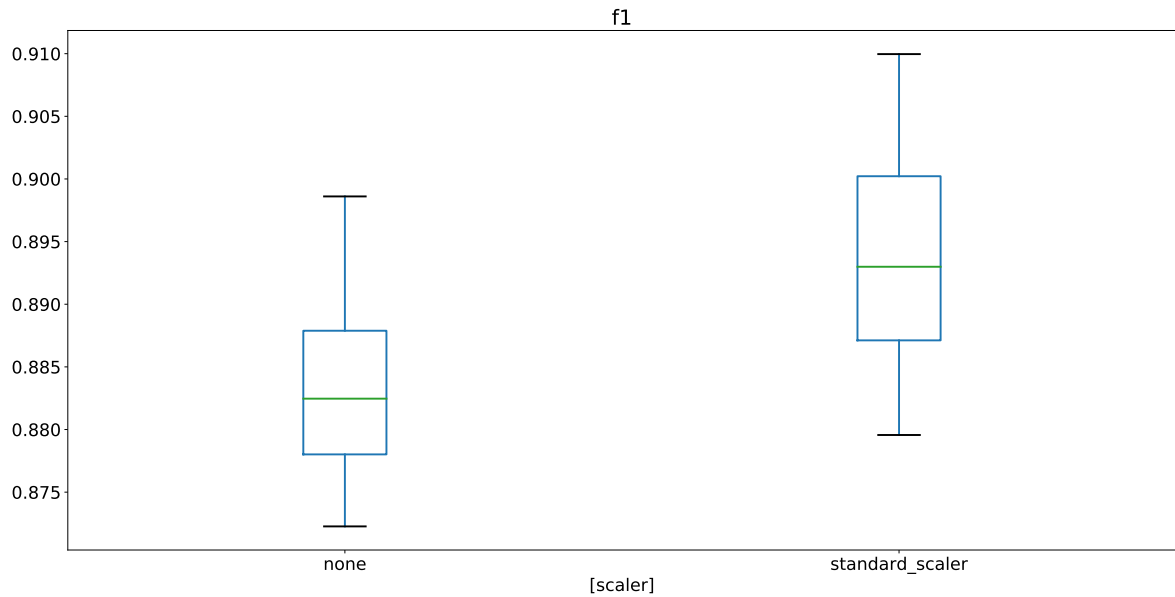


**Figure 6.2:** Graph showing the difference of score per network levels in the *S-wave*

## 6.2 Scaler

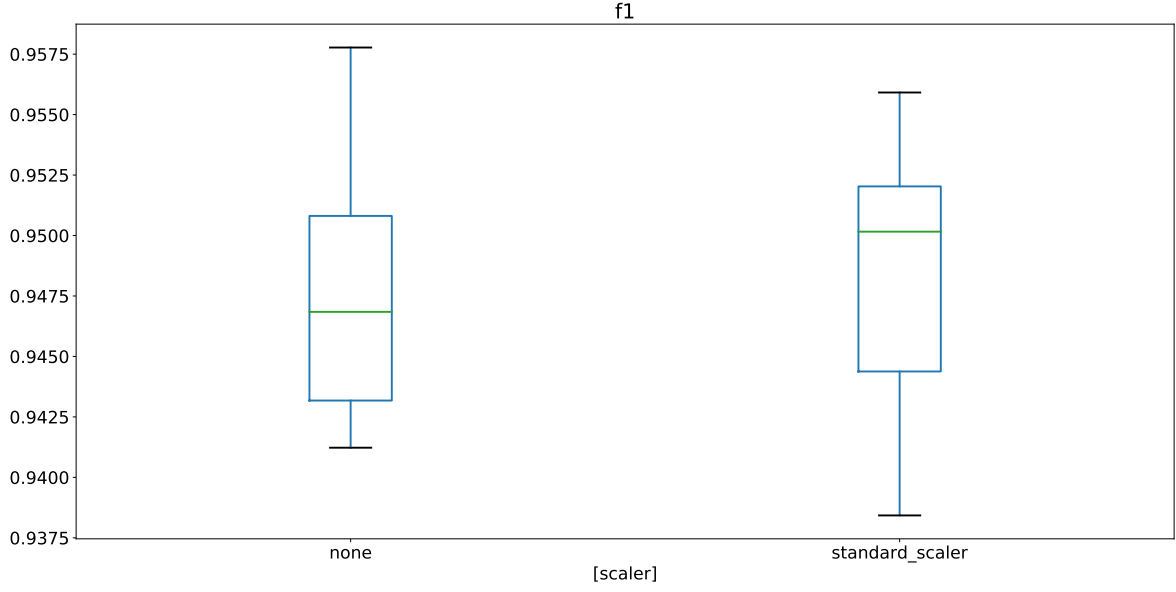
The scaler is another important factor for us, we wanted to see during our experiments if scaling the data before the training and prediction of our models could be helpful for improving the final F1 score.

In this case we got mixed signals, for the *P-wave* experiments, ANOVA results showed that it is significant, meaning that it made a difference, in the figure 6.3 we can see that in average regardless of other parameters the results using a standard scaler were able to outperform results without scaling the data.



**Figure 6.3:** Graph showing the difference of score per scaler levels in the *P-wave*

The results were different for the *S-wave* experiments, even when we can see that in the figure 6.4 the standard scaler is slightly better than not using a scaler ANOVA found that this factor does not have enough significance to show that they are different.

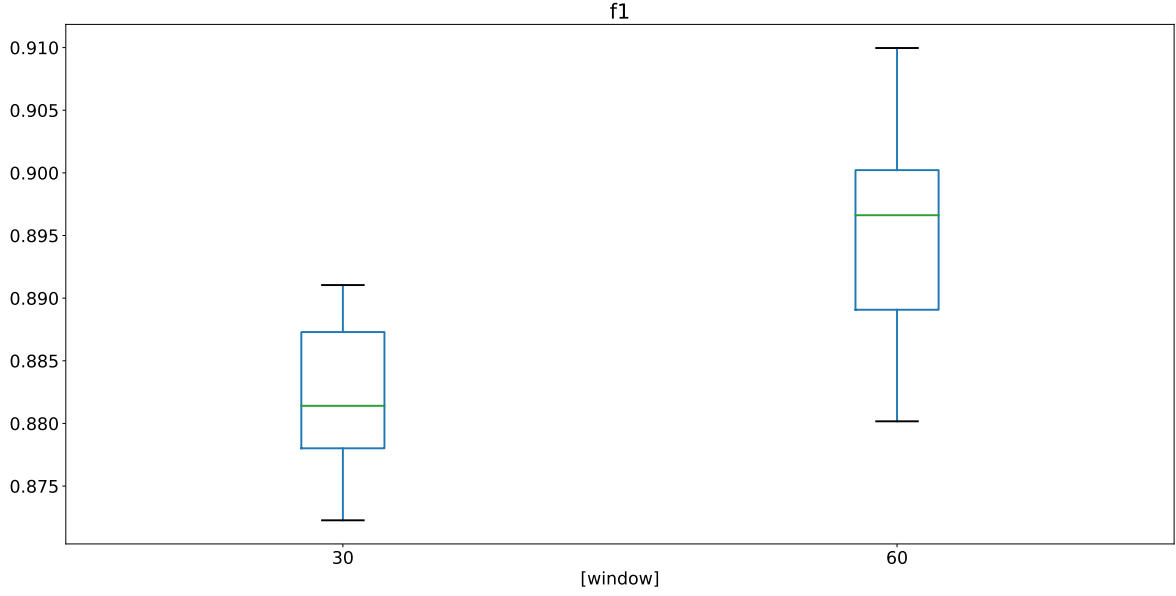


**Figure 6.4:** Graph showing the difference of score per scaler levels in the *S-wave*

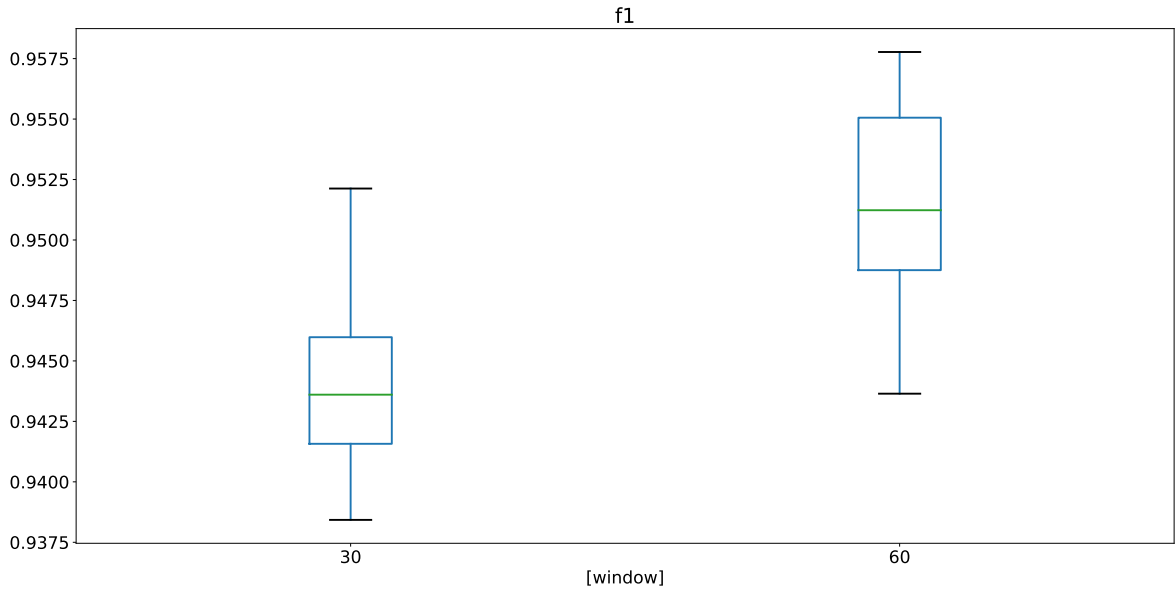
## 6.3 Window

ANOVA shows that the window is the most significant factor in both of our experiments. This is very important for us since it is a crucial part of our pre-processing steps. Our expectation was that bigger window sizes results in higher F1 scores, the experiments show that this was the case for this factor.

In the figures 6.5 and 6.6 we can see that the windows with a size of 60 is able to achieve a better result than a window of size 30 for both experiments.



**Figure 6.5:** Graph showing the difference of score per window levels in the *P-wave*



**Figure 6.6:** Graph showing the difference of score per window levels in the *S-wave*

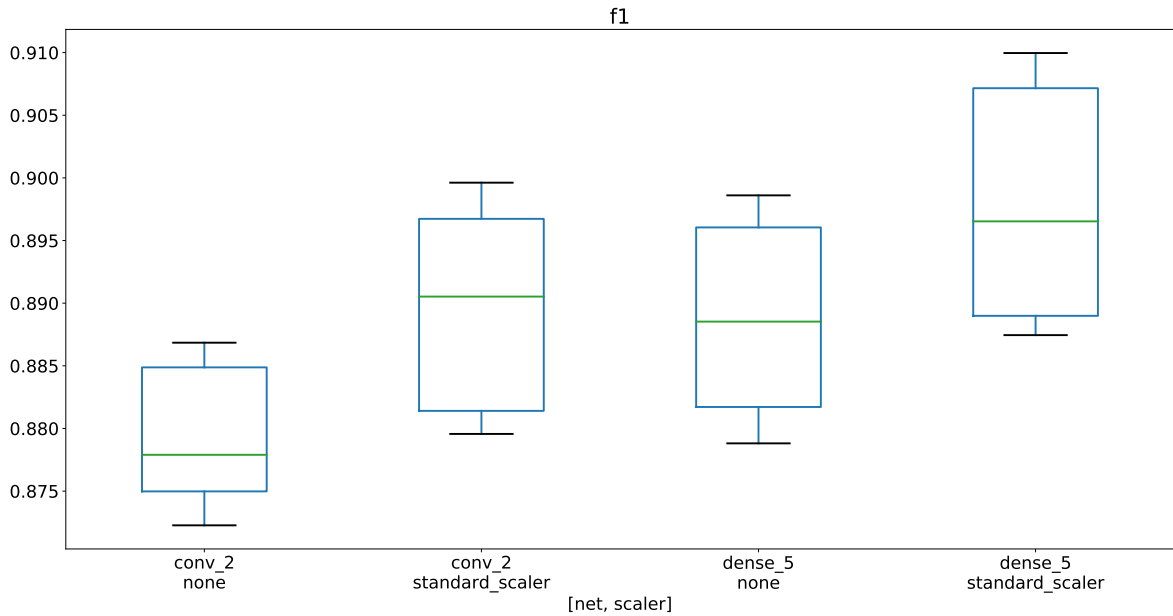
## 6.4 Network-Scaler

This interaction between Network and Scaler show us the effect of combining the network with different scalers. In the case of the *P-wave* experiments this was not a

significant factor.

Although that ANOVA result shows that interaction are not significant, Json C. Hsu in his book tell us that at Tukey test can show significant difference between this scenario [14].

We decided to look into it with a Tukey test, which tell us if there are difference between the means of at least two groups, as a whole ANOVA tell us that there's no difference between the groups, but there could be some hidden difference between a pair of those groups. From looking at the boxplot on figure 6.7 we can see that they perform similarly except for the Convolutional Neural Network without scaler.



**Figure 6.7:** Graph showing the difference of score per network and scaler levels in the *P-wave*

The listing 6.1 contains the Tukey test results, that shows that there is a difference between using the standard scaler and not using it for the Convolutional network. The other difference that we can find is between the pairs (conv\_2, none) and (dense.5,



standard\_scaler), however it is not clear if this is caused by the different model or the different scaler used.

```

1 Multiple Comparison of Means - Tukey HSD, FWER=0.05
2 =====
3           group1           group2           meandiff p-adj   lower  upper  reject
4 -----
5           (conv_2, none) (conv_2, standard_scaler)  0.0101 0.0363  0.0005 0.0197   True
6           (conv_2, none)           (dense_5, none)  0.0093 0.0595 -0.0003 0.0189  False
7           (conv_2, none) (dense_5, standard_scaler)  0.0186  0.001   0.009 0.0282   True
8 (conv_2, standard_scaler)           (dense_5, none) -0.0008   0.9 -0.0104 0.0088  False
9 (conv_2, standard_scaler) (dense_5, standard_scaler)  0.0085 0.0984 -0.0011 0.0181  False
10          (dense_5, none) (dense_5, standard_scaler)  0.0093 0.0622 -0.0003 0.0189  False
11 -----

```

**Listing 6.1:** Tukey - Network-Scaler – *P-wave*

Moving into the *S-wave* experiments ANOVA showed that there's a significant difference in this interaction, similarly to the previous analysis we decided to run a Tukey test and the results can be found in the listing 6.2.

```

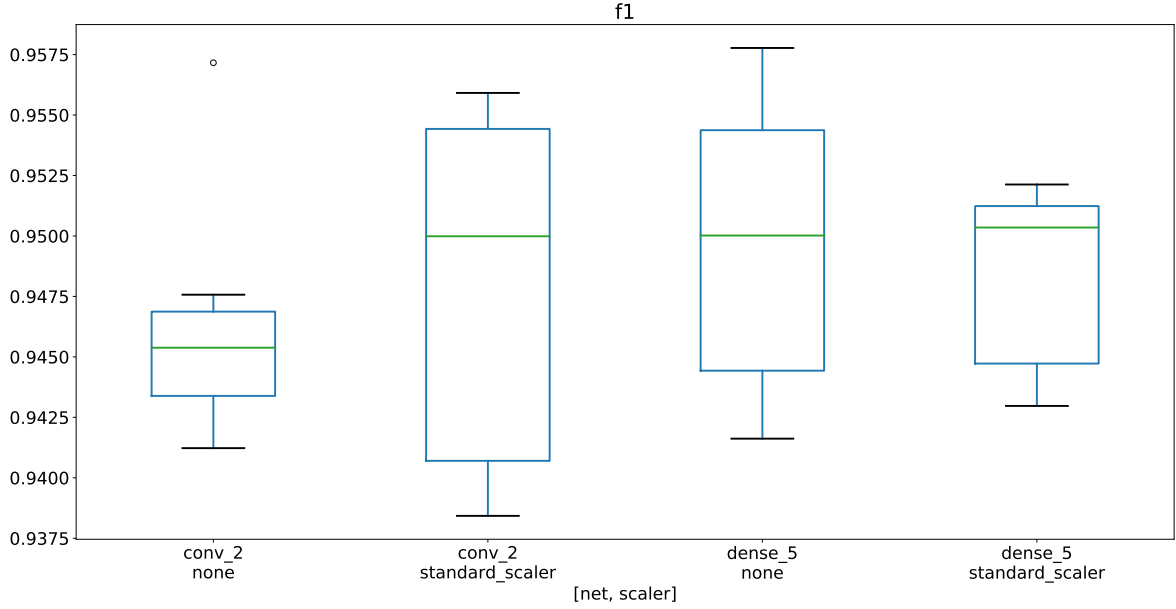
1 Multiple Comparison of Means - Tukey HSD, FWER=0.05
2 =====
3           group1           group2           meandiff p-adj   lower  upper  reject
4 -----
5           (conv_2, none) (conv_2, standard_scaler)  0.0027 0.5227 -0.0026  0.008  False
6           (conv_2, none)           (dense_5, none)  0.0043 0.1513 -0.001 0.0095  False
7           (conv_2, none) (dense_5, standard_scaler)  0.0026 0.5408 -0.0026 0.0079  False
8 (conv_2, standard_scaler)           (dense_5, none)  0.0016 0.8467 -0.0037 0.0068  False
9 (conv_2, standard_scaler) (dense_5, standard_scaler) -0.0001   0.9 -0.0053 0.0052  False
10          (dense_5, none) (dense_5, standard_scaler) -0.0016 0.8286 -0.0069 0.0036  False
11 -----

```

**Listing 6.2:** Tukey - Network-Scaler – *S-wave*

We can see that even when ANOVA told us that the interaction is significant, Tukey

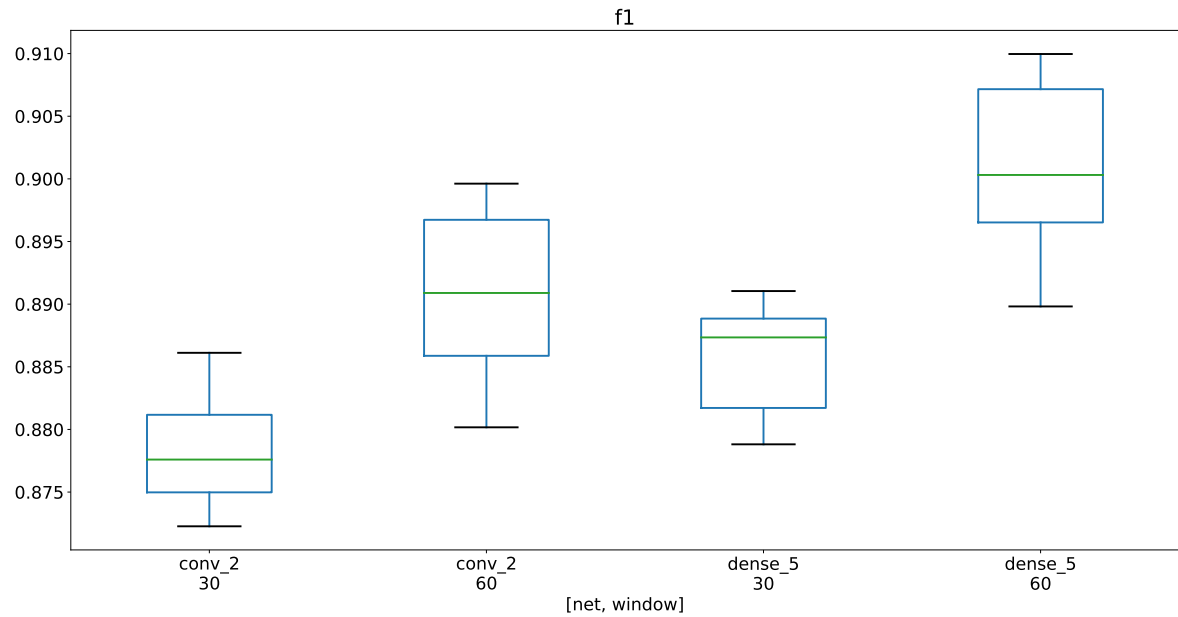
tell us that there's no evidence that each pair is different from each other, which also means that regardless of the network or scaler the models perform similarly for the *S-wave*. We can see the boxplot for this interaction in figure 6.8



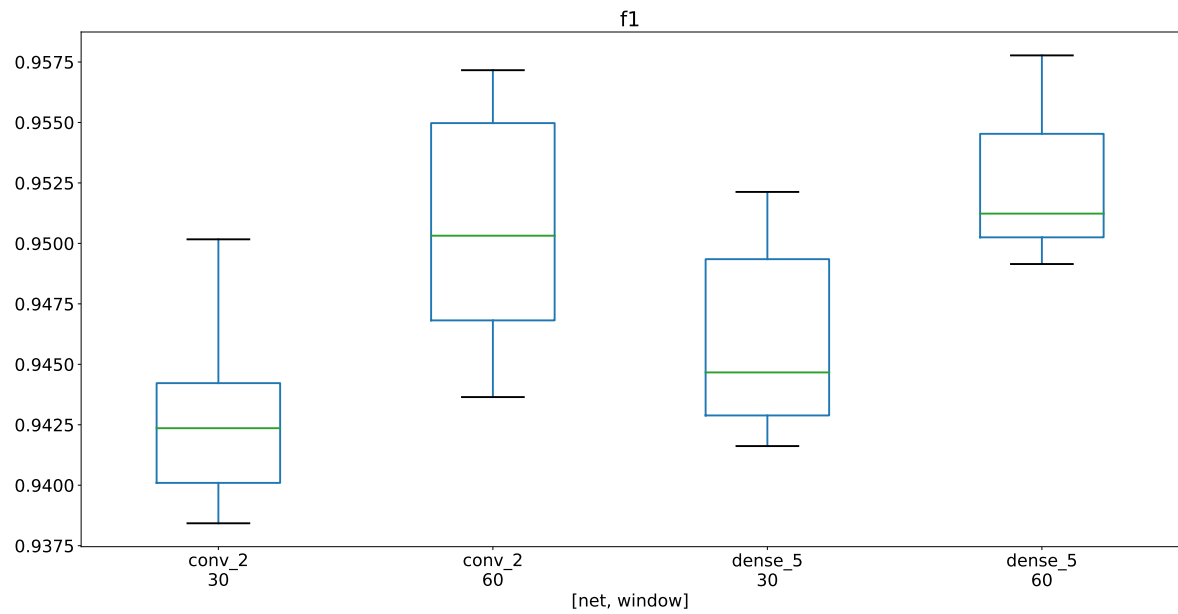
**Figure 6.8:** Graph showing the difference of score per network and scaler levels in the *S-wave*

## 6.5 Network-Window

For the Network and Window interaction, ANOVA shows that neither of the experiments (*P-wave* and *S-wave*) is significant. However to reference we attach the boxplot graphs (6.9, 6.10), but this information does not represent any statistical evidence.



**Figure 6.9:** Graph showing the difference of score per network and window levels in the *P-wave*



**Figure 6.10:** Graph showing the difference of score per network and window levels in the *S-wave*

## 6.6 Window-Scaler

The ANOVA results shows the effect of the interaction between Window and Scaler is a significant for the *P-wave*, however, we decided to run a Tukey test and the results can be found in the listing 6.3.

```

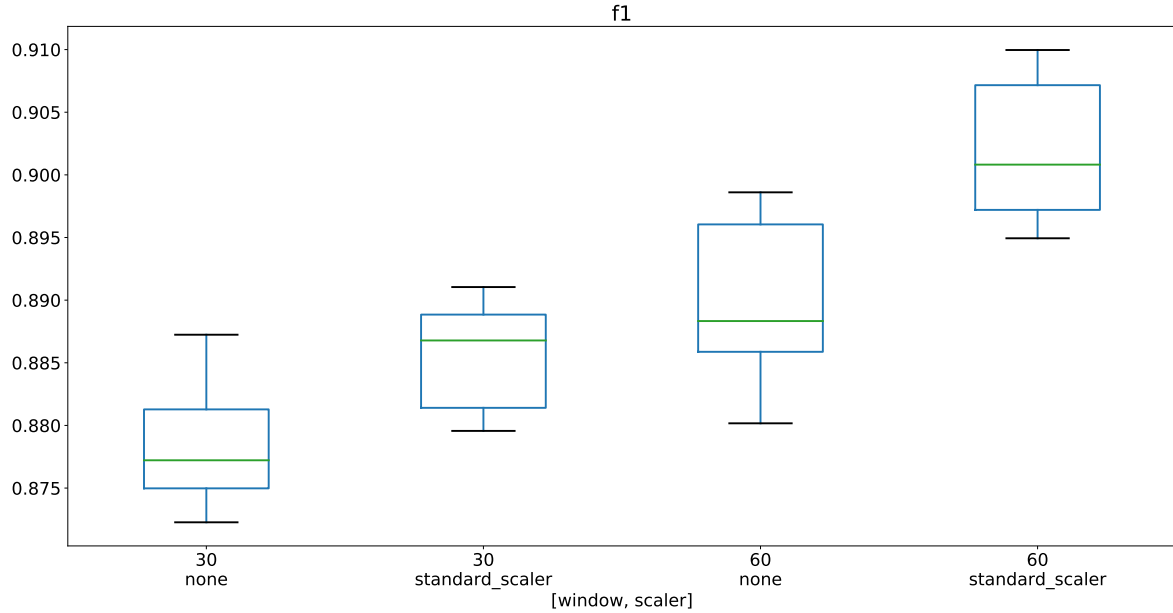
1 Multiple Comparison of Means - Tukey HSD, FWER=0.05
2 =====
3           group1           group2      meandiff p-adj  lower  upper  reject
4 -----
5           (30, none) (30, standard_scaler)  0.0072 0.0235 0.0008 0.0136   True
6           (30, none)           (60, none)  0.0116  0.001 0.0052 0.0181   True
7           (30, none) (60, standard_scaler)  0.0238  0.001 0.0174 0.0302   True
8 (30, standard_scaler)           (60, none)  0.0045 0.2605 -0.002 0.0109  False
9 (30, standard_scaler) (60, standard_scaler)  0.0166  0.001 0.0102  0.023   True
10          (60, none) (60, standard_scaler)  0.0122  0.001 0.0057 0.0186   True
11 -----

```

**Listing 6.3:** Tukey - Window-Scaler – *P-wave*

We can see that in Tukey test results that the window of size 30 and standard\_scaler has the same behavior that a window of size 60 without a scaler.

The boxplot graph in figure 6.11 shows that combining a window of size 60 with standard\_scaler improves the final result of F1 score, also we can see that the meadian of the f1 score for window of size 30 with standard scaler performs similarly to the window of size 60 without scaler, which is supported by the Tukey test results.



**Figure 6.11:** Graph showing the difference of score per window and scaler levels in the *P-wave*

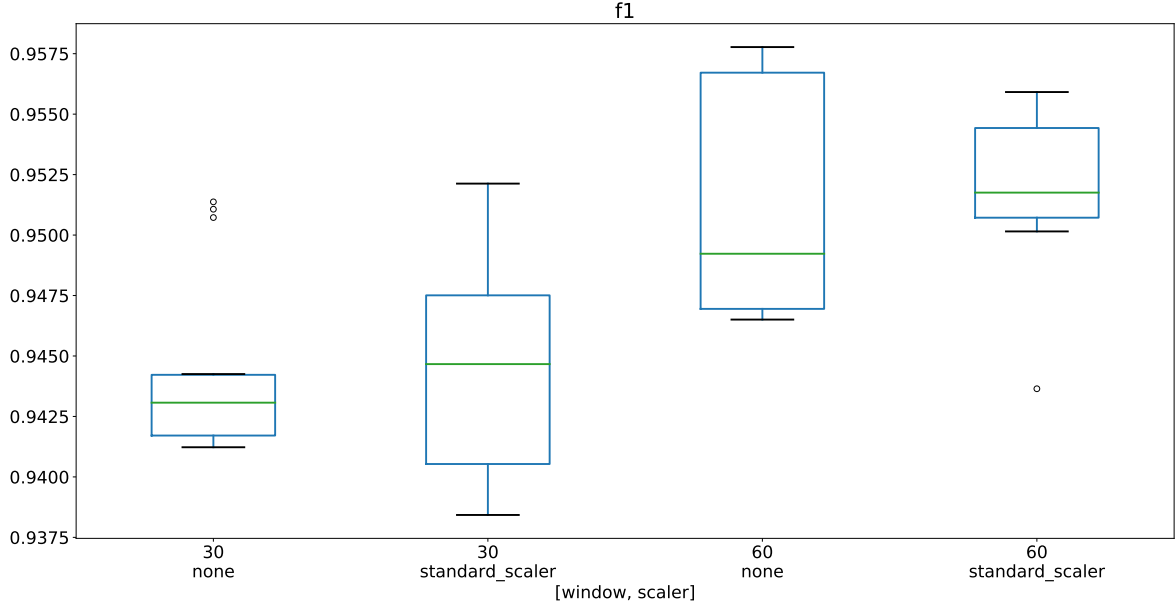
For the *S-wave* ANOVA shows that the Window and Scaler interaction was not a significant factor, anyway we decide to run a Tukey test, to obtain more statistic evidence 6.4, however the results are very similar to the results for *P-wave*.

We can see in the figure 6.12 that using a window of size 60 and standard\_scaler we get the best results, however, ANOVA does not support this claim.

```

1 Multiple Comparison of Means - Tukey HSD, FWER=0.05
2 =====
3      group1      group2      meandiff p-adj  lower  upper  reject
4 -----
5      (30, none) (30, standard_scaler)  0.0001  0.9 -0.0039 0.0042  False
6      (30, none)      (60, none)  0.0066 0.001  0.0026 0.0107  True
7      (30, none) (60, standard_scaler)  0.0076 0.001  0.0036 0.0117  True
8 (30, standard_scaler)      (60, none)  0.0065 0.001  0.0025 0.0106  True
9 (30, standard_scaler) (60, standard_scaler)  0.0075 0.001  0.0035 0.0116  True
10      (60, none) (60, standard_scaler)  0.001  0.9 -0.0031 0.005  False

```

**Listing 6.4:** Tukey - Window-Scaler – *S-wave***Figure 6.12:** Graph showing the difference of score per window and scaler levels in the *S-wave*

## 6.7 Network-Scaler-Window

The ANOVA result shows that the interaction that involve all of our factors is not significant. We decide to apply a Tukey test [14] to obtain statistical evidence that allow us to draw conclusiones about what is happening with the interaction of our three factors (6.5, 6.6.

In the figure 6.13 we can see that in the two networks the F1 score increases if we use the standar\_scaler and 60 as window size. We can also see that as we use better parameters found in previous experiments the Convolutional Neural Network is able to perform similarly to the Dense Network.

## 6.7. Network-Scaler-Window

1 Multiple Comparison of Means - Tukey HSD, FWER=0.05

=====							
group1	group2	meandiff	p-adj	lower	upper	reject	
-----							
(conv_2, 30, none)	(conv_2, 30, standard)	0.0074	0.0016	0.0022	0.0126	True	
(conv_2, 30, none)	(conv_2, 60, none)	0.01	0.001	0.0047	0.0152	True	
(conv_2, 30, none)	(conv_2, 60, standard)	0.0228	0.001	0.0175	0.028	True	
(conv_2, 30, none)	(dense_5, 30, none)	0.0077	0.001	0.0024	0.0129	True	
(conv_2, 30, none)	(dense_5, 30, standard)	0.0146	0.001	0.0094	0.0199	True	
(conv_2, 30, none)	(dense_5, 60, none)	0.021	0.001	0.0157	0.0262	True	
(conv_2, 30, none)	(dense_5, 60, standard)	0.0325	0.001	0.0273	0.0377	True	
(conv_2, 30, standard)	(conv_2, 60, none)	0.0026	0.715	-0.0026	0.0078	False	
(conv_2, 30, standard)	(conv_2, 60, standard)	0.0154	0.001	0.0102	0.0206	True	
(conv_2, 30, standard)	(dense_5, 30, none)	0.0003	0.9	-0.0049	0.0055	False	
(conv_2, 30, standard)	(dense_5, 30, standard)	0.0073	0.0018	0.0021	0.0125	True	
(conv_2, 30, standard)	(dense_5, 60, none)	0.0136	0.001	0.0084	0.0188	True	
(conv_2, 30, standard)	(dense_5, 60, standard)	0.0251	0.001	0.0199	0.0303	True	
(conv_2, 60, none)	(conv_2, 60, standard)	0.0128	0.001	0.0076	0.018	True	
(conv_2, 60, none)	(dense_5, 30, none)	-0.0023	0.8145	-0.0075	0.0029	False	
(conv_2, 60, none)	(dense_5, 30, standard)	0.0047	0.1051	-0.0005	0.0099	False	
(conv_2, 60, none)	(dense_5, 60, none)	0.011	0.001	0.0058	0.0162	True	
(conv_2, 60, none)	(dense_5, 60, standard)	0.0225	0.001	0.0173	0.0277	True	
(conv_2, 60, standard)	(dense_5, 30, none)	-0.0151	0.001	-0.0203	-0.0099	True	
(conv_2, 60, standard)	(dense_5, 30, standard)	-0.0081	0.001	-0.0133	-0.0029	True	
(conv_2, 60, standard)	(dense_5, 60, none)	-0.0018	0.9	-0.007	0.0034	False	
(conv_2, 60, standard)	(dense_5, 60, standard)	0.0097	0.001	0.0045	0.0149	True	
(dense_5, 30, none)	(dense_5, 30, standard)	0.007	0.003	0.0018	0.0122	True	
(dense_5, 30, none)	(dense_5, 60, none)	0.0133	0.001	0.0081	0.0185	True	
(dense_5, 30, none)	(dense_5, 60, standard)	0.0248	0.001	0.0196	0.03	True	
(dense_5, 30, standard)	(dense_5, 60, none)	0.0063	0.0093	0.0011	0.0115	True	
(dense_5, 30, standard)	(dense_5, 60, standard)	0.0178	0.001	0.0126	0.023	True	
(dense_5, 60, none)	(dense_5, 60, standard)	0.0115	0.001	0.0063	0.0167	True	

**Listing 6.5:** Tukey - Network-Window-Scaler – *P-wave*

```

1 Multiple Comparison of Means - Tukey HSD, FWER=0.05
2 =====
3      group1      group2      meandiff p-adj    lower    upper    reject
4 -----
5      (conv_2, 30, none) (conv_2, 30, standard)  0.0008    0.9 -0.0056  0.0072  False
6      (conv_2, 30, none)      (conv_2, 60, none)  0.0056 0.1278 -0.0008  0.012  False
7      (conv_2, 30, none) (conv_2, 60, standard)  0.0102 0.001  0.0038  0.0167  True
8      (conv_2, 30, none)      (dense_5, 30, none)  0.0032 0.7286 -0.0032  0.0097  False
9      (conv_2, 30, none) (dense_5, 30, standard)  0.0026 0.8918 -0.0038  0.0091  False
10     (conv_2, 30, none)      (dense_5, 60, none)  0.0109 0.001  0.0045  0.0173  True
11     (conv_2, 30, none) (dense_5, 60, standard)  0.0083 0.004  0.0018  0.0147  True
12     (conv_2, 30, standard)      (conv_2, 60, none)  0.0048 0.2805 -0.0016  0.0112  False
13     (conv_2, 30, standard) (conv_2, 60, standard)  0.0094 0.001  0.003  0.0158  True
14     (conv_2, 30, standard)      (dense_5, 30, none)  0.0024    0.9 -0.004  0.0088  False
15     (conv_2, 30, standard) (dense_5, 30, standard)  0.0018    0.9 -0.0046  0.0083  False
16     (conv_2, 30, standard)      (dense_5, 60, none)  0.0101 0.001  0.0037  0.0165  True
17     (conv_2, 30, standard) (dense_5, 60, standard)  0.0074 0.013  0.001  0.0139  True
18     (conv_2, 60, none) (conv_2, 60, standard)  0.0046 0.3297 -0.0018  0.011  False
19     (conv_2, 60, none)      (dense_5, 30, none) -0.0024    0.9 -0.0088  0.004  False
20     (conv_2, 60, none) (dense_5, 30, standard) -0.003 0.8005 -0.0094  0.0035  False
21     (conv_2, 60, none)      (dense_5, 60, none)  0.0053 0.1775 -0.0011  0.0117  False
22     (conv_2, 60, none) (dense_5, 60, standard)  0.0026 0.8918 -0.0038  0.0091  False
23     (conv_2, 60, standard)      (dense_5, 30, none) -0.007 0.024 -0.0134 -0.0006  True
24     (conv_2, 60, standard) (dense_5, 30, standard) -0.0076 0.0108 -0.014 -0.0011  True
25     (conv_2, 60, standard)      (dense_5, 60, none)  0.0007    0.9 -0.0057  0.0071  False
26     (conv_2, 60, standard) (dense_5, 60, standard) -0.002    0.9 -0.0084  0.0045  False
27     (dense_5, 30, none) (dense_5, 30, standard) -0.0006    0.9 -0.007  0.0059  False
28     (dense_5, 30, none)      (dense_5, 60, none)  0.0077 0.0093  0.0013  0.0141  True
29     (dense_5, 30, none) (dense_5, 60, standard)  0.005 0.2283 -0.0014  0.0115  False
30     (dense_5, 30, standard)      (dense_5, 60, none)  0.0083 0.004  0.0018  0.0147  True

```



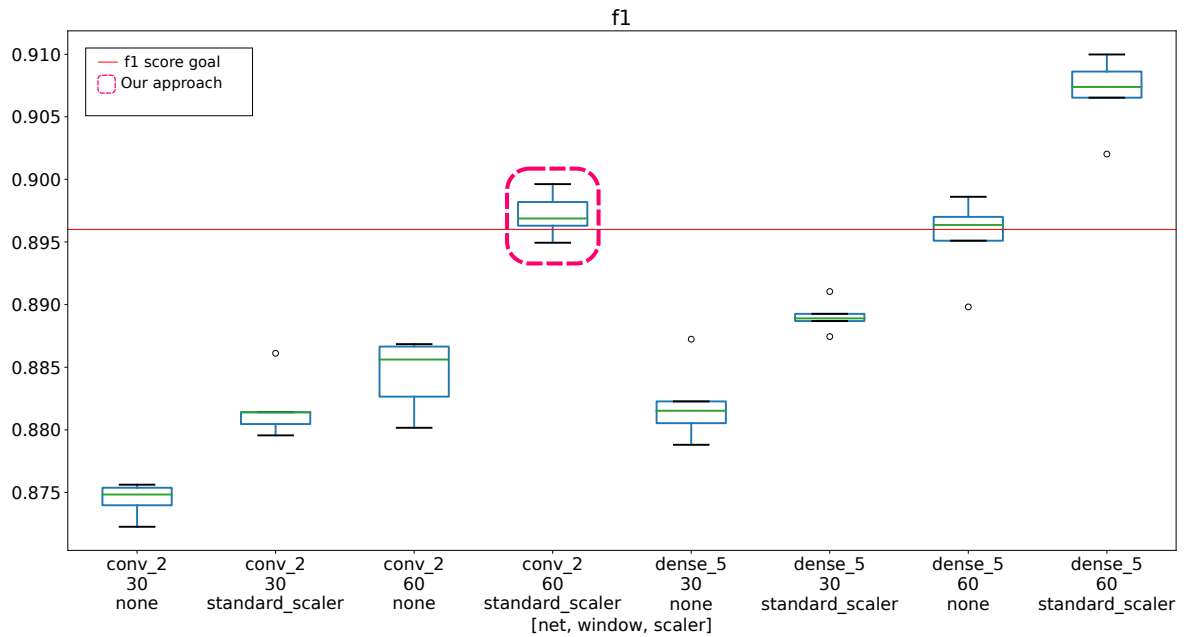
```

31 (dense_5, 30, standard) (dense_5, 60, standard)    0.0056 0.1278 -0.0008    0.012 False
32     (dense_5, 60, none) (dense_5, 60, standard)  -0.0027    0.89 -0.0091    0.0038 False
33 -----

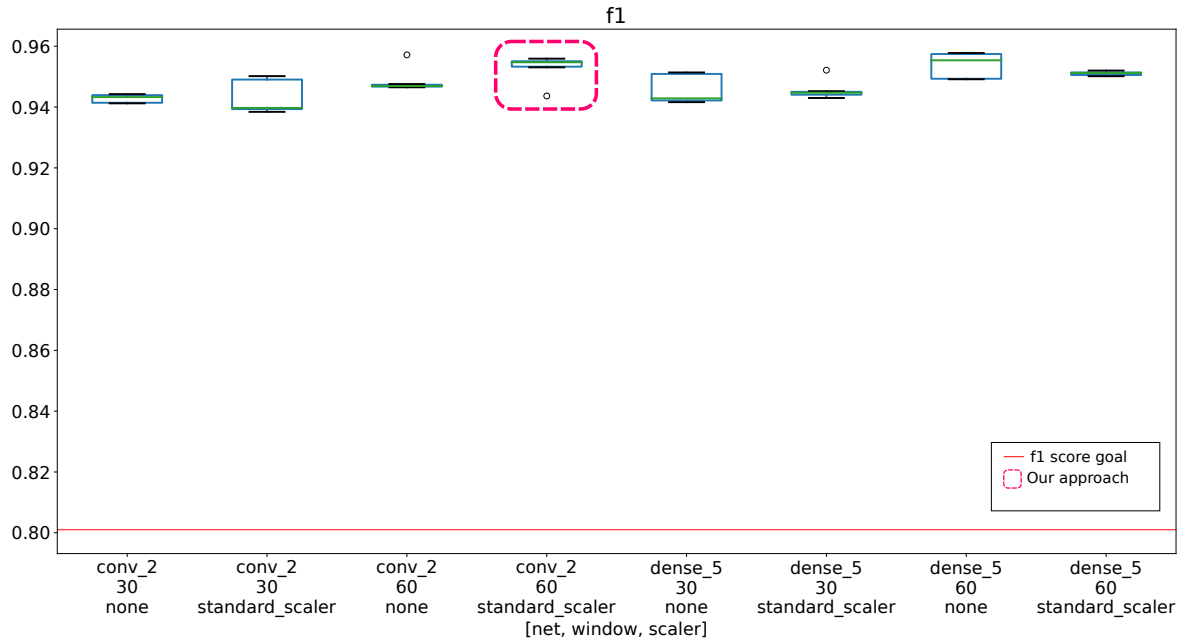
```

**Listing 6.6:** Tukey - Network-Window-Scaler – *S-wave*

Also in the figure 6.13 we indicate the F1 score goal for the *P-wave* that was defined in the hypothesis, we can observe that in general for the Convolutional Neural Network our hypothesis was not met in most of the cases, however, we have one particular case in which by using the best parameters for the scaler and window the hypothesis is true, here we conclude that our hypothesis is partially fulfilled.

**Figure 6.13:** Graph showing the difference of score per network, scaler and window levels in the *P-wave*

In the figure 6.14, we can see that a similar behavior as the *P-wave*, in the sense that as we use better parameters, the f1-score improves. In this figure 6.14 the red line also represents the F1 score goal for the S seismic wave. We can see the all combinations of Convolutional Neural Network are above the our goal, so we can confirm that this part of the hypothesis is true.



**Figure 6.14:** Graph showing the difference of score per network, scaler and window levels in the *S-wave*

## 6.8 Summary

In general, the hypothesis was met for the *S-wave* but missed the goal for the *P-wave* (See figure 6.1), because we have three *P-wave* cases in which alternative hypothesis is not true; see the table 6.2. However, in this seismic wave, we have one combination in which the alternative hypothesis met our goal. Also, we can see in table 6.3 that for the *S-wave* the alternative hypothesis is achieved on all the combinations.

We can see in the tables 6.2, 6.3 that our hypothesis partially fulfilled.

Phase	Mean F1 Score	Goal F1 Score	$H_1$	p_value
P	0.8844	0.896	✗	< 0.001
S	0.9469	0.801	✓	< 0.05

**Table 6.1:** Differences of Means Summary

Model	Window	Scaler	Mean F1 Score	$H_1(\leq)$
Conv_2	30	None	0.8744	✗
Conv_2	60	None	0.8843	✗
Conv_2	30	Standard_Scaler	0.8817	✗
Conv_2	60	Standard_Scaler	0.8971	✓

**Table 6.2:** Differences of Means Summary – *P-wave*

Model	Window	Scaler	Mean F1 Score	$H_1(\leq)$
Conv_2	30	None	0.9427	✓
Conv_2	60	None	0.9487	✓
Conv_2	30	Standard_Scaler	0.9435	✓
Conv_2	60	Standard_Scaler	0.9529	✓

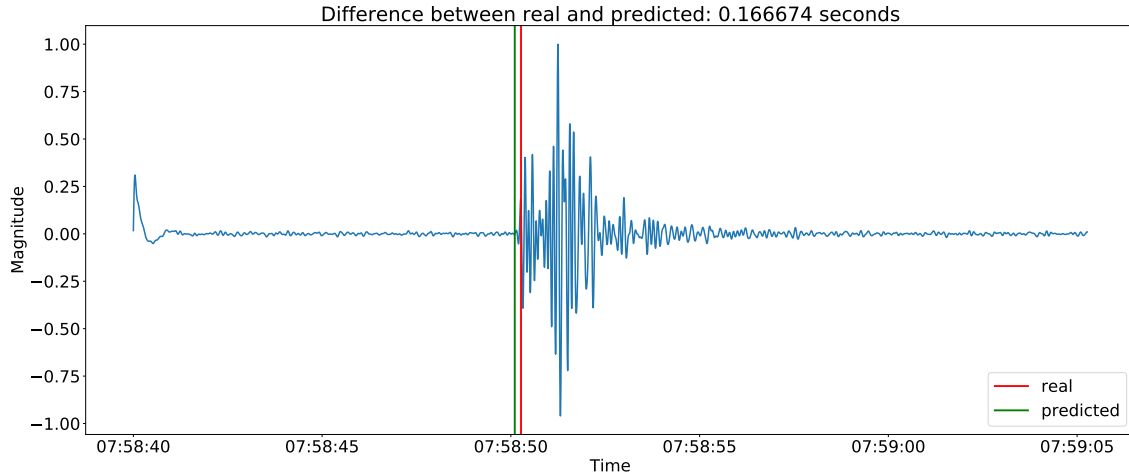
**Table 6.3:** Differences of Means Summary – *S-wave*

### 6.8.1 Final Results

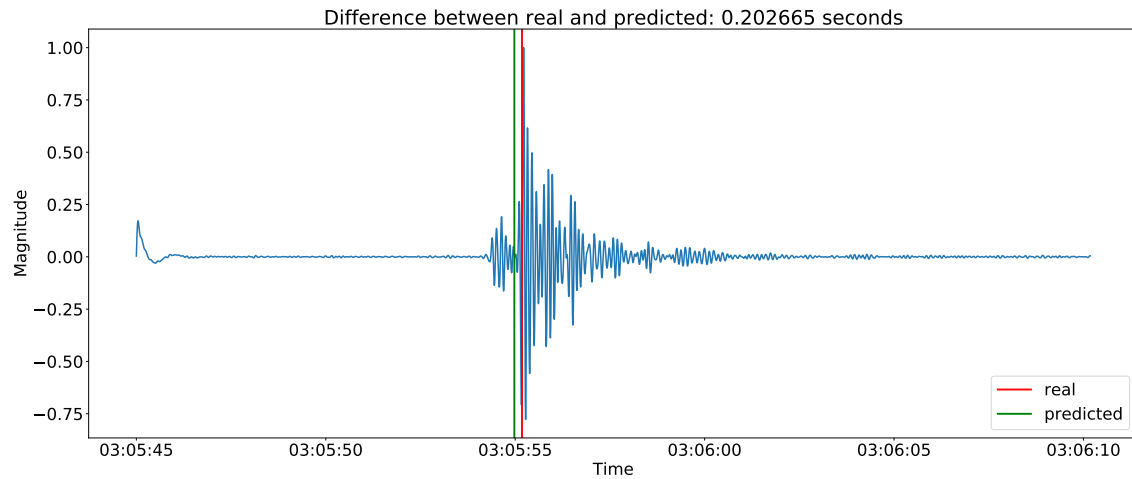
After all of the process was completed, including the calculation of the moving average, we calculated the difference in seconds between the human-calculated waves on the validation dataset and the prediction by our approach and calculated the absolute average error, the results are in the table 6.4, this was calculated using the code in the listing 5 in the annexes. Also the images below shows examples of how this looks in the seimograms, the images 6.15, 6.16, 6.17 shows examples for *P-wave* and the images 6.18, 6.19, 6.20 shows examples for the *S-wave*, this images were generated using the code in the listing 6 in the annexes.

<i>P-wave</i>	<i>S-wave</i>
<b>0.788</b> seconds	<b>0.35</b> seconds

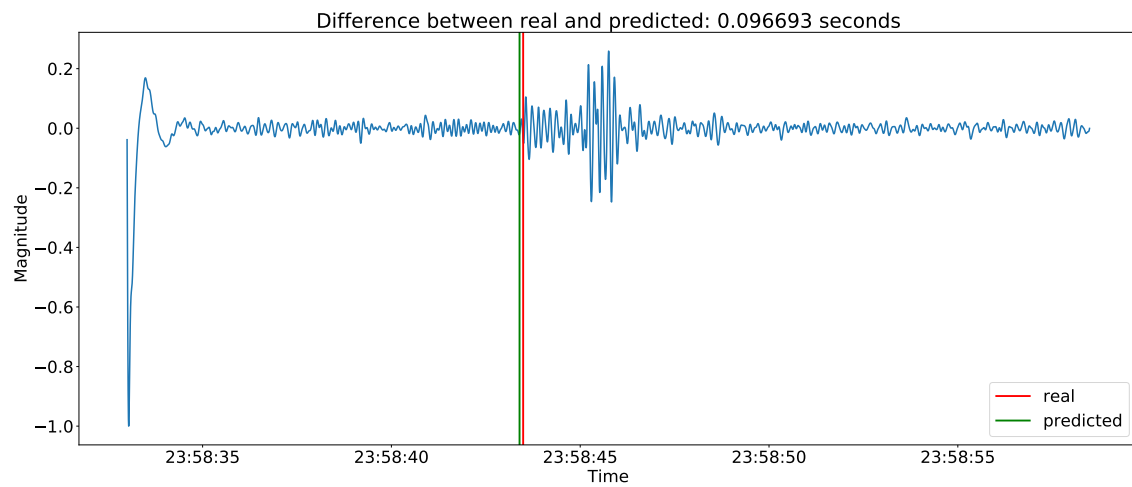
**Table 6.4:** Average of Differences Between Real and Predicted



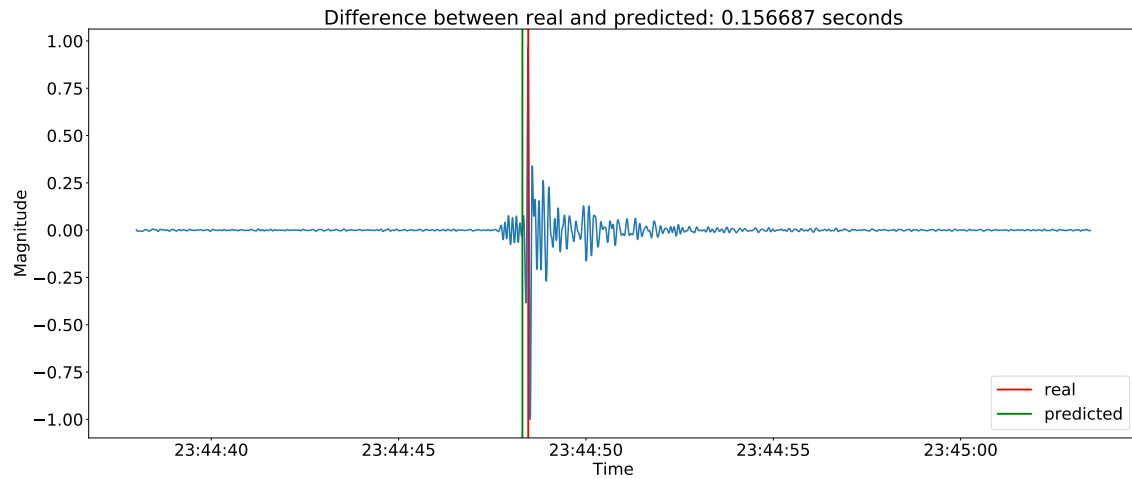
**Figure 6.15:** Difference seconds between real and predicted *P-wave-1*



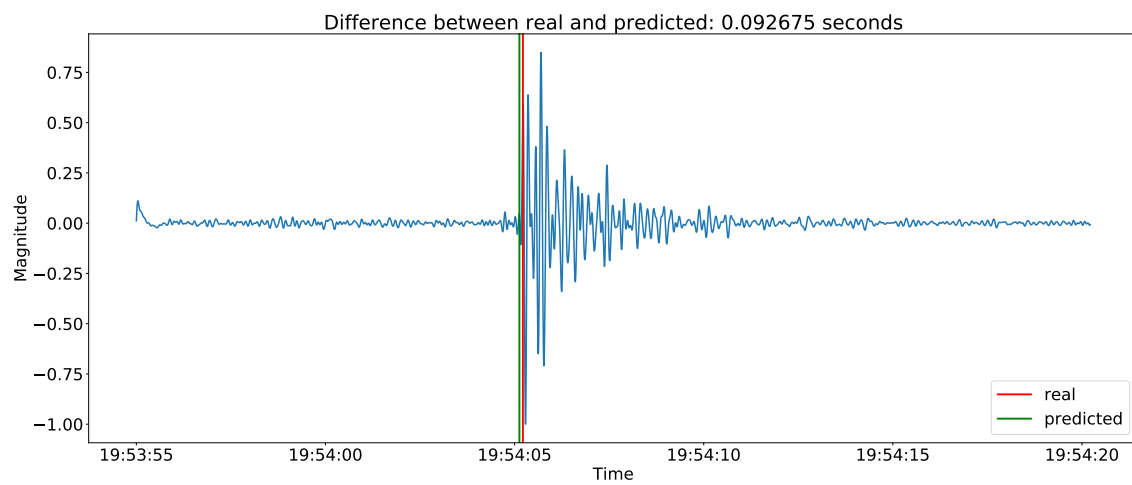
**Figure 6.16:** Difference seconds between real and predicted *P-wave-2*



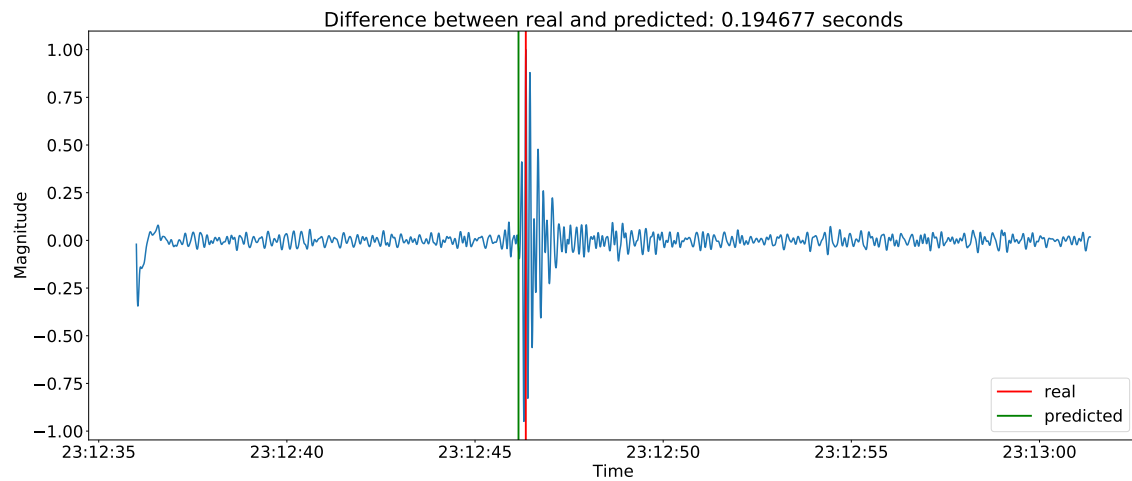
**Figure 6.17:** Difference seconds between real and predicted *P-wave-3*



**Figure 6.18:** Difference seconds between real and predicted *S-wave-1*



**Figure 6.19:** Difference seconds between real and predicted *S-wave-2*



**Figure 6.20:** Difference seconds between real and predicted *S-wave-3*

---

## Conclusions and Future Work

---

In this chapter, we will present the conclusions of this thesis and possible future research proposals that could improve the results presented here.

### 7.1 Conclusions

As the previous chapter discussed, we have several conclusions to present based on the research performed.

The first and what we think is the most important conclusion of this thesis is that there's evidence supporting the Convolutional Neural Network models are able to achieve in an at least one case the 0.986 F1 score primary waves and is able to achieve in all cases the 0.801 F1 score for secondary waves, which means that our hypothesis is partially true.

Another conclusion is that in general, Dense Neural Networks have a better F1



score than Convolutional Neural Networks, with the caveat that Dense Neural Networks are more intensive and have more parameters to train than the Convolutional Neural Networks, in our models the Dense Networks have around 9 time more trainable parameters than the Convolutional Networks.

The proposed pre-processing steps shows that by pre-processing the data we are able to get higher F1 score than using the raw wave data as input, an improvement of around 0.9% for the *P-wave* and 0.05% for the *S-wave* by using the `standard_scaler` and an improvement of 1.41% for the *P-wave* and 0.70% for the *S-wave* by using the correct window size.

## 7.2 Future Work

Based on the above conclusions, we propose the following paths to continue this research:

- Explore more options to pre-process data.
- Use the wave as a factor in the experiments.
- Use other size of windows.
- Explore other factors related to the seismic wave information to see their effects in the experiments (i.e. magnitude).
- Use dataset from different locations.
- Try out other machine learning algorithms with the pre-processing to look for possible improvements.
- Explore other data balancing methods.

## 7.3 Final Words

In general we believe that the research presented here was very successful, not only we were able to check the hypothesis and found some scenarios when the hypothesis is true, which was very satisfying, but also we were able to successfully comply with all of our objectives.

We discovered that the preprocessing of data improves the results, which we consider to be a great contribution from our research, which could potentially improve other techniques not only for seismographic time series but also any kind of time series that are being used in other researches.

We covered different topics of the Classification Models, such as Convolutional Neural Network and Dense Neural Network, also we were able to apply all of them to our research.

We also were able to identify possible improvements that can be applied to this research as future work, which we think will improve the precision to correctly identify the seismic waves.

---

## References

---

- [1] Mario Fernández A. and Javier Pacheco A. Sismotectónica de la región central de Costa Rica. *Revista Geológica de América Central*, 1998.
- [2] Takashi AKAZAWA. A technique for automatic detection of onset time of p and s-phases in strong motion records. *13th World Conference on Earthquake Engineering*, aug 2004.
- [3] Charles J. Ammon. Waves, seismograms, and seismometers, aug 2010. URL <http://eqseis.geosc.psu.edu/~cammon/HTML/Classes/IntroQuakes/Notes/seismometers.html#seismograms>. [Internet; consultado 10-noviembre-2018].
- [4] L. Braile. Seismic wave demonstrations and animations, feb 2010. URL [http://web.ics.purdue.edu/~braile/edumod/waves/WaveDemo.htm#Wave\\_Animations](http://web.ics.purdue.edu/~braile/edumod/waves/WaveDemo.htm#Wave_Animations). [Internet; consultado 9-noviembre-2018].
- [5] Jason Brownlee. *Clever Algorithms: Nature-Inspired Programming Recipes*. Lulu.com, 1st edition, 2011. ISBN 1446785068, 9781446785065.
- [6] Northern California Earthquake Data Center. Northern California Seismic Network, 2018. URL <http://www.ncedc.org/ncsn/>. [Internet; consultado 19-noviembre-2018].
- [7] Esteban J. Chaves, Marino Protti, and Cyrill Muller. The Costa Rica - OVSICORI - UNA geodynamic control network: A first world class network in a developing country. *IUGG19 Montreal, Canada.*, 2019.
- [8] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *MIC 2015: The XI Metaheuristics International Conference*, apr 2015. URL <https://arxiv.org/pdf/1502.02127.pdf>.
- [9] Gregory W Corder and Dale I Foreman. Nonparametric statistics for non-statisticians : a step-by-step approach. *Hoboken: John Wiley & Sons*, 2009.

- [10] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-scale convolutional neural networks for time series classification, 2016.
- [11] Sumit Das, Aritra dey, Akash Pal, and Nabamita Roy. Applications of artificial intelligence in machine learning: Review and prospect. *International Journal of Computer Applications*, 115:31–41, 04 2015. doi: 10.5120/20182-2402.
- [12] T. Dietterich. Machine learning for sequetial data: A review. *Oregon State University, Corvallis, Oregon, USA*, 2002.
- [13] Sergio Morales Esquivel. Evaluación de métodos agregados de aprendizaje de máquina: Aplicación sobre datos de la misión kepler. Master’s thesis, Ingeniería en Computación, Tecnológico de Costa Rica, oct 2016.
- [14] Jason C. Hsu. *Multiple Comparison Theory and Methods*. Guilford School Practitioner, 1996.
- [15] Q. Huang, M. Gerstenberger, and J Zhuang. Current challenges in statistical seismology. *Pure and Applied Geophysics*, 2016.
- [16] LeNet – Convolutional Neural Network in Python. Adrian rosebrock, aug 2016. URL <https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/>.
- [17] IRIS. How often do earthquakes occur? *Education and Outreach Series*, 2011.
- [18] K. M. Keranen, H. M. Savage, G. A. Abers, and E. S. Cochran. Potentially induced earthquakes in Oklahoma, USA: Links between wastewater injection and the 2011 mw 5.7 earthquake sequence. *Geology* 41, 41(6):699–702, jun 2013.
- [19] Suki Lau. A walkthrough of convolutional neural network—hyperparameter tuning. Towards Data Science, jul 2017. URL <https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd>.
- [20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [21] Evan Lutin. Grid searching in machine learning: Quick explanation and python implementation. Medium, sep 2017. URL <https://medium.com/@elutins/grid-searching-in-machine-learning-quick-explanation-and-python-implementation-550552200596>.
- [22] Craig Mertler and Rachel Vannatta Reinhart. *Advanced and multivariate statistical methods: Practical application and interpretation: Sixth edition*. Taylor and Francis, 10 2016. ISBN 9781138289734. doi: 10.4324/9781315266978.
- [23] J. B. Mockus and L. J. Mockus. Bayesian approach to global optimization and application to multiobjective and constrained problems. *J. Optim. Theory Appl.*, 70(1):157–172, July 1991. ISSN 0022-3239. doi: 10.1007/BF00940509. URL <https://doi.org/10.1007/BF00940509>.

- [24] Mary Natrella. *NIST/SEMATECH e-Handbook of Statistical Methods*. NIST/SEMATECH, July 2010. URL <http://www.itl.nist.gov/div898/handbook/>.
- [25] Michael A. Nielsen. *Neural networks and deep learning*, 2018. URL <http://neuralnetworksanddeeplearning.com/>.
- [26] Chawla N.V. Data mining for imbalanced datasets: An overview. In: *Maimon O., Rokach L. (eds) Data Mining and Knowledge Discovery Handbook*. Springer, 2009.
- [27] University of California Berkeley. Northern california earthquake data center. URL <https://ncedc.org>.
- [28] OVSICORI-UNA. Acerca del instituto, 2015. URL <http://www.ovsicori.una.ac.cr/index.php/ovsicori/acerca-instituto>. [Internet; consultado 10-octubre-2018].
- [29] OVSICORI-UNA. Sismicidad histórica, 2015. URL <http://www.ovsicori.una.ac.cr/index.php/sismologia>. [Internet; consultado 10-octubre-2018].
- [30] OVSICORI-UNA. Placas tectónicas, 2016. URL <http://www.ovsicori.una.ac.cr/index.php/extension/seccion-educativa>. [Internet; consultado 20-octubre-2018].
- [31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, J. Vanderplas V. Dubourg, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python - compare the effect of different scalers on data with outliers. *Journal of Machine Learning Research*, pages 2825–2830, 2011.
- [32] Thibaut Perol, Michaël Gharbi, and Marine Denolle. Convolutional neural network for earthquake detection and location. *Science Advances*, feb 2018.
- [33] Earthquake Hazards Program. Earthquake glossary. URL <https://earthquake.usgs.gov/learn/glossary/>. [Internet; consultado 10-noviembre-2018].
- [34] R Project. What is r, 2013. URL <http://www.r-project.org/about.html>. [Internet; consultado 10-octubre-2018].
- [35] Shashank Ramesh. A guide to an efficient way to build neural network architectures- part i: Hyper-parameter selection and tuning for dense networks using hyperas on fashion-mnist. Towards Data Science, may 2018.
- [36] Shashank Ramesh. A guide to an efficient way to build neural network architectures- part ii: Hyper-parameter selection and tuning for dense networks using hyperas on fashion-mnist. Towards Data Science, may 2018.
- [37] Mohammed Rampurawala. Classification with tensorflow and dense neural networks, feb 2019. URL <https://heartbeat.fritz.ai/classification-with-tensorflow-and-dense-neural-networks-829932>.

- [38] Giovanni Sanabria Brenes. *Comprendiendo la estadística inferencial*. Editorial Tecnológica de Costa Rica, 2011. ISBN 78-9977-66-238-1.
- [39] Giovanna Sannino and Giuseppe De Pietro. A deep learning approach for ecg-based heartbeat classification for arrhythmia detection. *Future Generation Computer Systems*, 86:446–455, sep 2018.
- [40] D. P. Schaff, G. H. R. Bokelmann, G. C. Beroza, and F. Waldhauser and W. L. Ellsworth. Highresolution image of calaveras fault seismicity. *Journal of Geophysical Research Solid Earth*, 107:ESE 5–1–ESE 5–16, sep 2002.
- [41] Rajesh S. Brid Grey Atom School. Time series, dec 2018. URL <https://medium.com/greyatom/time-series-b6ef79c27d31>.
- [42] Peter M. Shearer. *Introduction to Seismology*. Cambridge, second edition, jul 2009.
- [43] Koo Ping Shung. Accuracy, Precision, Recall or F1? Towards Data Science, mar 2015. URL <https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9>.
- [44] Red Sismológica Nacional UCR-ICE. Historia, aug 2019. URL <https://rsn.ucr.ac.cr/rsn/historia>.
- [45] Cleto González Víquez. *Temblores, terremotos, inundaciones, y erupciones volcánicas en Costa Rica, 1608-1910*. Editorial Tecnológica de Costa Rica, 1994.
- [46] R.E. Walpole, R.H. Myers, and S.L. Myers. *Probability and Statistics for Engineers and Scientists*. Pearson Education, Limited, 2010. ISBN 9780321629111. URL <http://books.google.co.cr/books?id=tzZxRQAACAAJ>.
- [47] Wikipedia. Machine learning, 2018. URL [http://www.wikiwand.com/en/Machine\\_learning](http://www.wikiwand.com/en/Machine_learning). [Internet; consultado 10-noviembre-2018].
- [48] Wikipedia. Time series, 2018. URL [http://www.wikiwand.com/en/Time\\_series](http://www.wikiwand.com/en/Time_series). [Internet; consultado 10-noviembre-2018].
- [49] Michele M. Wood. What is seismology?, may 2017. URL <http://www.geo.mtu.edu/UPSeis/waves.html>. [Internet; consultado 9-noviembre-2018].
- [50] Observatorio Vulcanológico y Sismológico de Costa Rica. Catálogo sísmico. *Universidad Nacional. Costa Rica*, 2017-2018. doi: 10.5281/zenodo.1478247.
- [51] Clara E. Yoon, Ossian O’Reilly, Karianne J. Bergen, and Gregory C. Beroza. Earthquake detection through computationally efficient similarity search. *Science Advances*, dec 2015.
- [52] Weiqiang Zhu and Gregory C. Beroza. Phasenet: A deep-neural-network-based seismic arrival time picking method. mar 2018. URL <https://arxiv.org/abs/1803.03211>.

## Acrónimos

***AI*** Artificial Intelligence

***ANOVA*** Analysis of Variance

***AR-AIC*** Auto Regression- Akaike Information Criterion

***ConvNet*** Convolutional Neural Network

***DoE*** Design of Experiments

***FAST*** Fingerprint And Similarity Thresholding

***LTA*** Long-Term-Average

***P*** primary wave

***S*** secondary wave

***STA*** Short-Term-Average

## Python code used to Analyze the data

```
1 p_metrics = pd.read_csv(p_conv2_dense5_processed.csv).set_index([net, window, scaler])
2
3 # Shapiro Test
4 shapiro_result = stats.shapiro(p_metrics[f1])[1]
5
6 #Levene Test
7 populations = []
8 for net in p_metrics.reset_index()[net].unique():
9     for window in p_metrics.reset_index()[window].unique():
10         for scaler in p_metrics.reset_index()[scaler].unique():
11             populations.append(p_metrics.reset_index()
12                               .query(f"net=={net}&window=={window}&scaler=={scaler}")[f1])
13 levene_result = stats.levene(*populations)[1]
14
15 # Anova Test
16 model = ols(f1~net*window*scaler, data=p_metrics.reset_index()).fit()
17 aov_table = sm.stats.anova_lm(model, typ=2)
18
19 # Residuals Test
20 residuals = model.resid
21 fitted = model.fittedvalues
22 smoothed = lowess(residuals,fitted)
23 top3 = abs(residuals).sort_values(ascending = False)[:3]
24
25 #Tukey tests
26 p_metrics_reset = p_metrics.reset_index().drop([net], axis=1)
27 p_metrics_reset = p_metrics_reset.set_index([window, scaler])
28 mc = MultiComparison(p_metrics_reset.reset_index()[f1],
29                      [str(group) for group in p_metrics_reset.index])
30 window_scaler_result = mc.tukeyhsd()
31
32 p_metrics_reset = p_metrics.reset_index().drop([window], axis=1)
33 p_metrics_reset = p_metrics_reset.set_index([net, scaler])
34 mc = MultiComparison(p_metrics_reset.reset_index()[f1],
35                      [str(group) for group in p_metrics_reset.index])
36 net_scaler_result = mc.tukeyhsd()
```



---

```

37
38 p_metrics_reset = p_metrics.reset_index().drop([scaler], axis=1)
39 p_metrics_reset = p_metrics_reset.set_index([net, window])
40 mc = MultiComparison(p_metrics_reset.reset_index()[f1],
41                      [str(group) for group in p_metrics_reset.index])
42 net_window_result = mc.tukeyhsd()
43
44 p_metrics_reset = p_metrics.reset_index()
45 p_metrics_reset.loc[p_metrics_reset[scaler] == standard_scaler, scaler] = standard
46 p_metrics_reset = p_metrics_reset.set_index([net, window, scaler])
47 mc = MultiComparison(p_metrics_reset.reset_index()[f1],
48                      [str(group) for group in p_metrics_reset.index])
49 net_window_scaler_result = mc.tukeyhsd()
50
51 #Metrics
52 scaler_metrics = p_metrics.reset_index()
53 (scaler_metrics[scaler_metrics[scaler] == standard_scaler][f1]
54  .mean() - scaler_metrics[scaler_metrics[scaler] == none][f1].mean())*100
55 window_metrics = p_metrics.reset_index()
56 (window_metrics[window_metrics[window] == 30][f1]
57  .mean() - window_metrics[window_metrics[window] == 60][f1].mean())*100
58 conv = p_metrics.reset_index()
59 conv = conv[conv[net] == conv_2]
60 conv = conv.groupby([net]).agg(np.mean)
61 conv[h0] = conv[f1] > 0.896

```

Listing 1: Python Code - Analyze *P-wave*

```

1 s_metrics = pd.read_csv(s_metrics_processed.csv).set_index([net, window, scaler])
2
3 # Shapiro Test
4 shapiro_result = stats.shapiro(s_metrics[f1])[1]
5
6
7 #Levene Test
8 populations = []
9 for net in s_metrics.reset_index()[net].unique():
10     for window in s_metrics.reset_index()[window].unique():
11         for scaler in s_metrics.reset_index()[scaler].unique():
12             populations.append(s_metrics.reset_index()
13                               .query(f"net_==_{net}_&_scaler_==_{scaler}_&_window_==_{window}")
14                               [f1])
15 levene_result = stats.levene(*populations)[1]
16
17 # Anova Test
18 model = ols(f1_~_net_*_window_*_scaler, data=s_metrics.reset_index()).fit()
19 aov_table = sm.stats.anova_lm(model, typ=2)
20
21 # Residuals Test
22 residuals = model.resid
23 fitted = model.fittedvalues
24 smoothed = lowess(residuals, fitted)
25 top3 = abs(residuals).sort_values(ascending = False)[:3]
26
27 #Tukey tests
28 s_metrics_reset = s_metrics.reset_index().drop([net], axis=1)

```

---

```

28 s_metrics_reset = s_metrics_reset.set_index([window, scaler])
29
30 mc = MultiComparison(s_metrics_reset.reset_index()[f1],
31                      [str(group) for group in s_metrics_reset.index])
32 window_scaler = mc.tukeyhsd()
33
34 s_metrics_reset = s_metrics_reset.reset_index().drop([window], axis=1)
35 s_metrics_reset = s_metrics_reset.set_index([net, scaler])
36
37 mc = MultiComparison(s_metrics_reset.reset_index()[f1],
38                      [str(group) for group in s_metrics_reset.index])
39 net_scaler_result = mc.tukeyhsd()
40
41 s_metrics_reset = s_metrics_reset.reset_index().drop([scaler], axis=1)
42 s_metrics_reset = s_metrics_reset.set_index([net, window])
43
44 mc = MultiComparison(s_metrics_reset.reset_index()[f1],
45                      [str(group) for group in s_metrics_reset.index])
46 net_window_result = mc.tukeyhsd()
47
48 s_metrics_reset = s_metrics_reset.reset_index()
49 s_metrics_reset = s_metrics_reset.set_index([net, window, scaler])
50
51 mc = MultiComparison(s_metrics_reset.reset_index()[f1],
52                      [str(group) for group in s_metrics_reset.index])
53 net_window_scaler_result = mc.tukeyhsd()
54
55 #Metrics
56 scaler_metrics = s_metrics.reset_index()
57 (scaler_metrics[scaler_metrics[scaler] == standard_scaler][f1]
58  .mean() - scaler_metrics[scaler_metrics[scaler] == none][f1].mean())*100
59 window_metrics = s_metrics.reset_index()
60 (window_metrics>window_metrics>window == 30)[f1]
61  .mean() - window_metrics>window_metrics>window == 60)[f1].mean())*100
62 conv = s_metrics.reset_index()
63 conv = conv[conv[net] == conv_2]
64 conv = conv.groupby([net]).agg(np.mean)
65 conv[h0] = conv[f1] > 0.801

```

Listing 2: Python Code - Analyze *S-wave*

## Experiments *P-wave* Results

```

1 net>window>scaler>f1
2 conv_2,30,none,0.875624971540458
3 conv_2,30,none,0.8753837072620708
4 conv_2,30,none,0.8748415647806332
5 conv_2,30,none,0.8739827468596791
6 conv_2,30,none,0.8722672866016118
7 conv_2,30,standard_scaler,0.8861147128287378
8 conv_2,30,standard_scaler,0.8814104415417241
9 conv_2,30,standard_scaler,0.8814016908381084
10 conv_2,30,standard_scaler,0.8804663706202102
11 conv_2,30,standard_scaler,0.8795627230282481

```

---

```

12 conv_2,60,none,0.8868454526719936
13 conv_2,60,none,0.8866526284377864
14 conv_2,60,none,0.8856169525978967
15 conv_2,60,none,0.8826528846069391
16 conv_2,60,none,0.8801684445049723
17 conv_2,60,standard_scaler,0.8996182435845022
18 conv_2,60,standard_scaler,0.8981877439648311
19 conv_2,60,standard_scaler,0.8968765740159946
20 conv_2,60,standard_scaler,0.8962984557064896
21 conv_2,60,standard_scaler,0.8949380586210782
22 dense_5,30,none,0.8872405672784437
23 dense_5,30,none,0.8822738006175861
24 dense_5,30,none,0.8815279674833708
25 dense_5,30,none,0.8805360110532189
26 dense_5,30,none,0.878810895182295
27 dense_5,30,standard_scaler,0.8910444875285458
28 dense_5,30,standard_scaler,0.8892636008102944
29 dense_5,30,standard_scaler,0.8889002178631104
30 dense_5,30,standard_scaler,0.888687918425881
31 dense_5,30,standard_scaler,0.8874497650413583
32 dense_5,60,none,0.898606011932079
33 dense_5,60,none,0.897003160819211
34 dense_5,60,none,0.8963615892999336
35 dense_5,60,none,0.8951001743167201
36 dense_5,60,none,0.8898173990957496
37 dense_5,60,standard_scaler,0.90996604543369
38 dense_5,60,standard_scaler,0.9086027557810488
39 dense_5,60,standard_scaler,0.9073732130918296
40 dense_5,60,standard_scaler,0.9065165975241244
41 dense_5,60,standard_scaler,0.902017215180462

```

Listing 3: Experiments *P-wave* Results

## Experiments *S-wave* Results

```

1 net>window,scaler,f1
2 conv_2,30,none,0.9442523170563007
3 conv_2,30,none,0.9441283290319842
4 conv_2,30,none,0.943699692944406
5 conv_2,30,none,0.9432828458681524
6 conv_2,30,none,0.9414328895654784
7 conv_2,30,none,0.9413916482480352
8 conv_2,30,none,0.9412264271670719
9 conv_2,30,standard_scaler,0.9501681852894712
10 conv_2,30,standard_scaler,0.9498069836045484
11 conv_2,30,standard_scaler,0.9482713185702576
12 conv_2,30,standard_scaler,0.9397204855262372
13 conv_2,30,standard_scaler,0.9395664320166102
14 conv_2,30,standard_scaler,0.9391154107935474
15 conv_2,30,standard_scaler,0.9384262840628784
16 conv_2,60,none,0.9571616060310106
17 conv_2,60,none,0.9475710372150484
18 conv_2,60,none,0.947095459029925
19 conv_2,60,none,0.9468985165306826

```

---

```

20 conv_2,60,none,0.9467886337665682
21 conv_2,60,none,0.9466875043462666
22 conv_2,60,none,0.9465057278880928
23 conv_2,60,standard_scaler,0.9559140498664824
24 conv_2,60,standard_scaler,0.9550702476295482
25 conv_2,60,standard_scaler,0.9550526278816546
26 conv_2,60,standard_scaler,0.9547356564374652
27 conv_2,60,standard_scaler,0.9535004967973983
28 conv_2,60,standard_scaler,0.9530643768826152
29 conv_2,60,standard_scaler,0.9436452136289888
30 dense_5,30,none,0.9513730723823632
31 dense_5,30,none,0.9510690330203933
32 dense_5,30,none,0.95072442461353
33 dense_5,30,none,0.9428566388955849
34 dense_5,30,none,0.9423520119769656
35 dense_5,30,none,0.9419848469963062
36 dense_5,30,none,0.9416189085357496
37 dense_5,30,standard_scaler,0.9521276751707212
38 dense_5,30,standard_scaler,0.9452164547266104
39 dense_5,30,standard_scaler,0.9447753184795312
40 dense_5,30,standard_scaler,0.9447059152187752
41 dense_5,30,standard_scaler,0.9446228023754925
42 dense_5,30,standard_scaler,0.9435084241823588
43 dense_5,30,standard_scaler,0.9429718471235748
44 dense_5,60,none,0.9577733957371096
45 dense_5,60,none,0.9574599192413288
46 dense_5,60,none,0.9574037554408044
47 dense_5,60,none,0.955374351154422
48 dense_5,60,none,0.9493128144004392
49 dense_5,60,none,0.949308620160764
50 dense_5,60,none,0.9491465219825904
51 dense_5,60,standard_scaler,0.9519991054271733
52 dense_5,60,standard_scaler,0.9515118584330714
53 dense_5,60,standard_scaler,0.9512385770306646
54 dense_5,60,standard_scaler,0.951223454163708
55 dense_5,60,standard_scaler,0.9505496847709608
56 dense_5,60,standard_scaler,0.9505483738886288
57 dense_5,60,standard_scaler,0.9501507016874308

```

Listing 4: Experiments *S-wave* Results

## Code for Final Results average calculation

```

1
2
3 from obspy import read
4 import matplotlib.pyplot as plt
5 from obspy.core.utcdatetime import UTCDateTime
6 import pandas as pd
7
8 from sklearn.preprocessing import StandardScaler
9 import numpy as np
10 from keras.models import load_model
11 import datetime as dt

```

---

```

12
13
14 import matplotlib.pyplot as plt
15 plt.rcParams.update({'font.size': 22})
16
17 def rolling_window(a, window, step_size, padding=True, copy=True):
18     if copy:
19         result = a.copy()
20     else:
21         result = a
22     if padding:
23         result = np.hstack((result, np.zeros(window)))
24     shape = result.shape[:-1] + (result.shape[-1] - window + 1 - step_size, window)
25     strides = result.strides + (result.strides[-1] * step_size,)
26     return np.lib.stride_tricks.as_strided(result, shape=shape, strides=strides)
27
28 def nearest(items, pivot):
29     return min(items, key=lambda x: abs(x - pivot))
30
31 def get_diff(mseed, model, real_p):
32     tr = read(mseed)[0]
33     tr.filter(bandpass, freqmin=1.0, freqmax=10.0, corners=4, zerophase=True)
34     tr.normalize()
35
36     timestamps = pd.Series([str(tr.stats.starttime + (((tr.stats.endtime - tr.stats.starttime)/len
37     timestamps = pd.to_datetime(timestamps)
38
39     real_p = pd.to_datetime(real_p)
40     nearest_data_point_real_p = nearest(timestamps, real_p)
41     df = pd.DataFrame({
42         timestamps: timestamps,
43         data: pd.Series(tr.data)
44     })
45
46     df.set_index(timestamps, inplace=True)
47     x = np.array(df[data])
48     # Scale data
49     scaler = StandardScaler()
50     scaler = scaler.fit(x.reshape(-1, 1))
51     x = scaler.transform(x.reshape(1, -1))[0]
52
53     # Window data
54     x = rolling_window(x, 60, 1)
55     x_dataset = np.reshape(x, (len(x), 60, 1))
56
57     results = model.predict(x_dataset)
58     p_probs = results[:, 1]
59     N=60
60     p_probs_mean = np.convolve(p_probs, np.ones((N,))/N, mode=valid)
61     guess = np.argmax(p_probs_mean)
62     predicted_p = df.iloc[guess].name
63
64     return abs(predicted_p - real_p).total_seconds()
65

```

---

```

66
67 p_files = []
68
69 import os
70 from os import walk
71
72 for (dirpath, dirnames, filenames) in walk(/validation/csv_data):
73     for file in filenames:
74         p_files.append(/validation/ + file)
75     break
76
77
78 diff = 0
79 total = 0
80 model = load_model(model_to_evaluate.h5)
81
82 for file in p_files:
83     df = pd.read_csv(file)
84     df = df[df[phase] == S]
85     for mseed, timestamp in zip(df[waveform], df[time]):
86         mseed_file = os.path.join(/.join(file.split(/)[: -2]), mseed)
87         print(fprocessing: {mseed_file})
88         diff = diff + get_diff(mseed_file, model, timestamp)
89         total = total + 1
90
91 print(faverage_diff: {diff/total})

```

Listing 5: Average difference calculation

## Code for Final Graphs Generation

```

1
2 from obspy import read
3 import matplotlib.pyplot as plt
4 from obspy.core.utcdatetime import UTCDateTime
5 import pandas as pd
6
7 from sklearn.preprocessing import StandardScaler
8 import numpy as np
9 from keras.models import load_model
10 import datetime as dt
11
12
13 import matplotlib.pyplot as plt
14 plt.rcParams.update({font.size: 22})
15
16 def rolling_window(a, window, step_size, padding=True, copy=True):
17     if copy:
18         result = a.copy()
19     else:
20         result = a
21     if padding:
22         result = np.hstack((result, np.zeros(window)))
23     shape = result.shape[: -1] + (result.shape[-1] - window + 1 - step_size, window)

```

---

```

24     strides = result.strides + (result.strides[-1] * step_size,)
25     return np.lib.stride_tricks.as_strided(result, shape=shape, strides=strides)
26
27 def nearest(items, pivot):
28     return min(items, key=lambda x: abs(x - pivot))
29
30 def predict_and_plot(mseed, network, real_p):
31     tr = read(mseed)[0]
32     tr.filter(bandpass, freqmin=1.0, freqmax=10.0, corners=4, zerophase=True)
33     tr.normalize()
34
35     timestamps = pd.Series([str(tr.stats.starttime + (((tr.stats.endtime - tr.stats.starttime)/len
36     timestamps = pd.to_datetime(timestamps)
37
38     real_p = pd.to_datetime(real_p)
39     nearest_data_point_real_p = nearest(timestamps, real_p)
40     df = pd.DataFrame({
41         timestamps: timestamps,
42         data: pd.Series(tr.data)
43     })
44
45     df.set_index(timestamps, inplace=True)
46     x = np.array(df[data])
47     # Scale data
48     scaler = StandardScaler()
49     scaler = scaler.fit(x.reshape(-1, 1))
50     x = scaler.transform(x.reshape(1, -1))[0]
51
52     # Window data
53     x = rolling_window(x, 60, 1)
54     x_dataset = np.reshape(x, (len(x), 60, 1))
55
56
57     model = load_model(network)
58
59     results = model.predict(x_dataset)
60     p_probs = results[:, 1]
61     N=60
62     p_probs_mean = np.convolve(p_probs, np.ones((N,))/N, mode=valid)
63     guess = np.argmax(p_probs_mean)
64     predicted_p = df.iloc[guess].name
65
66     plt.figure(figsize=(25,10))
67     plt.plot(df[real_p-dt.timedelta(seconds=15):real_p+dt.timedelta(seconds=15)], linewidth=2)
68     plt.axvline(nearest_data_point_real_p, color=red, linewidth=2.5, label=real)
69     plt.axvline(predicted_p, color=green, linewidth=2.5, label=predicted)
70     plt.title(fDifference between real and predicted: {abs(predicted_p-real_p).total_seconds()})
71     plt.legend(loc=lower_right)
72     plt.xlabel(Time)
73     plt.ylabel(Magnitude)
74     plt.savefig(f"{.join(mseed.split('.')[:-1])}.pdf", transparent=True, bbox_inches=tight, pad_inches=0.5)
75
76 predict_and_plot(BG_RGP_DPZ_2014-01-01T23:58:33_2014-01-01T23:59:03.mseed,
77                 conv_2__60__standard_scaler__False__0.9044293295860291__0.2368850384235382__426d7

```

---

```
78         2014-01-01T23:58:43.490000Z)
79
80 predict_and_plot(BG_HVC_DPE_2014-01-01T03:05:45_2014-01-01T03:06:15.mseed,
81                  conv_2__60__standard_scaler__False__0.9044293295860291__0.2368850384235382__426d7
82                  2014-01-01T03:05:55.180000Z)
83
84 predict_and_plot(BG_BRP_DPZ_2014-01-01T07:58:40_2014-01-01T07:59:10.mseed,
85                  conv_2__60__standard_scaler__False__0.9044293295860291__0.2368850384235382__426d7
86                  2014-01-01T07:58:50.270000Z)
87
88 predict_and_plot(BG_ACR_DPN_2014-03-01T23:44:38_2014-03-01T23:45:08.mseed,
89                  conv_2__60__standard_scaler__False__0.9520879883766175__1.7420735006837723e+32__3
90                  2014-03-01T23:44:48.460000Z)
91
92 predict_and_plot(BG_FNF_DPE_2014-03-01T23:12:36_2014-03-01T23:13:06.mseed,
93                  conv_2__60__standard_scaler__False__0.9520879883766175__1.7420735006837723e+32__3
94                  2014-03-01T23:12:46.350000Z)
95
96 predict_and_plot(BG_CLV_DPE_2014-03-01T19:53:55_2014-03-01T19:54:25.mseed,
97                  conv_2__60__standard_scaler__False__0.9520879883766175__1.7420735006837723e+32__3
98                  2014-03-01T19:54:05.220000Z)
```

**Listing 6:** Final Results Generation